Journal of Computer Science 2 (11): 824-834, 2006
ISSN 1549-3636

Mobile Agents in Heterogeneous Networks: A Look on Performance

R. B. Patel and Neeraj Goel
Department of Computer Engineering, M. M. Engineering College, Mullana-133203, Haryana, India

Abstract: Mobile agents can be seen as a new paradigm for distributed computing. One characteristic of mobile agents is their autonomy and pro-activity. This article present a framework designed on top of PMADE (Platform for Mobile Agent Distribution and Execution) to support our research in this domain. Our framework consists of three modules. The Map Module collects information on the available network of agent hosts (AHs) and a mobile agent is able to locate services within the network with the help of this module. Furthermore, the framework provides a Route estimating module and an optimization module, which minimizes network load. These modules support a fast and efficient navigation through the network, which is an environment with an inherently dynamical behavior. The basis for this work is the PMADE Domain Service, which is a hierarchical arrangement of AHs that groups AHs into a federation of networked local clusters, so called domains. This paper also evaluates the basic concepts and the actual modules of the developed framework and made some measurements to characterize the modules quality and to identify problems.

Key words: PMADE, mobile agents, route estimator, domain

INTRODUCTION

A Mobile agent (MA)[1] is a software process, which can move autonomously from one physical network location to another. The agent performs its job wherever and whenever it is found appropriate and is not restricted to be co-located with its client. Thus, there is an inherent sense of autonomy in the mobility and execution of the agent. Agents can be seen as automated errand boys who work for users. MA research evolved over the past years from the creation of many different monolithic mobile agent systems (MASs), often with similar characteristics and built by research groups spread all over the world, for optimisation and better understanding of specific agent issues[1,2].

In the area 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[3]. PMADE (Platform for Mobile Agent Distribution and Execution) is a Java-based platform that supports the efficient migration of mobile agents on a wide variety of protocols and migration strategies[4]. It is based on so called agent host (AH)[4] that resides on the network's nodes. In our approach, every Java-enabled device in the Internet can be such a network node. Currently, we are working on additional system components on top of the basic platform layer to network mobile AH better, to improve scalability and flexibility and to provide an information base for mobile agents that support their pro-activity and adaptability. Especially interesting is the case where the network provides a dynamic environment[5], e.g., if mobile network nodes and services appear and disappear and where agents act as intelligent entities by determining their own path at runtime dynamically in the continuously changing landscape.

The migration of mobile agents is associated with different movement costs viz transmission time, round trip time, number of hops, etc. Costs are also generated by executing the agent's algorithms on the AH. Such execution costs are not part of this research. The focus is to optimize the autonomous navigation through a network in general. We do not care for the agent's autonomy on the user task level, e.g., negotiations, fulfill user tasks, etc. We also do not optimize the technical (hardware) infrastructure of the underlying network. The movement of mobile agents is based on a logical network view which is based on the nodes on the network which are equipped with AH.

To improve the performance of mobile agents means to optimize a mobile agent's path through a network of AH (nodes). Thereby, an agent visits only those AHs which provide a service of interest. Furthermore, the agent uses a fast path through a network based on known infrastructure characteristics (as QoS). Finally, an agent optimizes its transmissions between AHs with the help of several migration strategies described in[6]. The main focus of this paper is on the performance and evaluation of the additional system components which are introduced in Section 3. All components have been prototyped and are currently evaluated. We look at their interaction, trace typical scenarios and discuss their efficiency in a number of situations. Based on the results of our experiments, we

Corresponding Author: R. B. Patel, Department of Computer Engineering, M. M. Engineering College, Mullana-133203, Haryana, India

will discuss the applicability of the proposed framework and pinpoint relevant advantages as well as inherent constraints.

Overview of PMADE: Figure 1 shows the basic block diagram of PMADE. Each node of the network has an Agent Host (AH), which is responsible for accepting and executing incoming autonomous Java agents and an Agent Submitter (AS)[5,6], which submits the MA on behalf of the user to the AH.

A user, who wants to perform a task, submits the MA designed to perform that task, to the AS on the user system. The AS then tries to establish a connection with the specified AH, where the user already holds an account. If the connection is established, the AS submits the MA to it and then goes offline. The AH examines the nature of the received agent and executes it. The execution of the agent depends on its nature and state. The agent can be transferred from one AH to another whenever required. On completion of execution, the agent submits its results to the AH, which in turn stores the results until the remote AS retrieves them for the user.

The AH is the key component of PMADE. It consists of the manager modules and the Host Driver. The Host Driver lies at the base of the PMADE architecture and the manager modules reside above it. It is the basic utility module responsible for driving the AH by ensuring proper co-ordination between various managers and making them work in tandem. Details of the managers and their functions are provided in[6]. PMADE provides weak mobility to its agents and allows one-hop, two-hop and multi-hop agents[7,8].

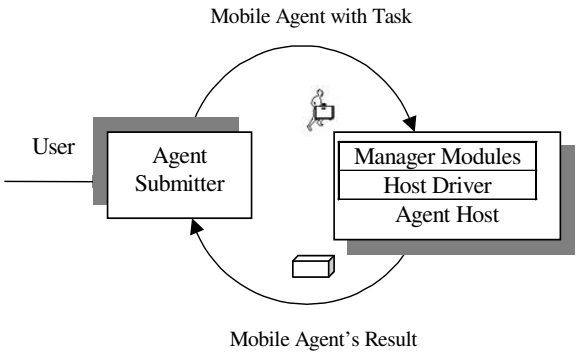

Fig. 1: Block architecture of PMADE

System architecture: The PMADE Domain Service is a hierarchical arrangement of groups of AHs into a federation of networked local clusters, so called domains is shown in Fig. 2. A domain is simply a local group of well connected and/or logically neighbored AH with a dedicated manager (a specialized AH-the Domain Manager[7]). Domains, holding only a limited number of nodes are again networked with other domains via a so-called master, thus providing the means to structure very large networks in a scalable and iterative fashion. The Domain Master is a specialized Domain Manager which manages only Domain Managers.

Building on this as an infrastructure, we collect 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[9]. The basic concepts of this module are taken from the Network Weather Service[10]. In addition to throughput, latency and other network status information, this module collects and distributes information on application-level services provided by the AH in the domain. By partitioning the network in PMADE Domains, each AH is located within a local domain that reflects its primary area of interest. Each Domain Manager has additional links to a number of selected remote domains. The Map Module cares for precise and up-to-date knowledge (maps) within its local domain and provides a rough, summarized view of the linked remote domains.

Utilizing the service descriptions in those maps, a mobile agent is able to locate points of interest within the network and see changes in the network structure. Once a list of interesting AH has been determined, another system component - the Route Estimator as shown in Fig. 3 - can be used by the agent to plan an itinerary[8]. This component is able to calculate the shortest trip through the net based on the map data. This component uses classic local optimization algorithms. If necessary, an itinerary can be recalculated and amended, for example, in the case of changes in the network or when the agent moves into new domains and thus shifts its focus with regards to the fisheye paradigm.

At any point in time, as long as we have an itinerary, a mobile agent may also use a so called Migration Planner as shown in Fig. 3 to optimize each single migration included in this itinerary from a more technological, in our case PMADE-specific, efficiency perspective[8]. 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 AH. This is, if necessary, done by a concept called slicing[11]. Other options are to place code in advance in the network, to send data home to carry fewer luggages, to change the transmission protocol, etc.

Figure 4 presents an architectural overview of the system. The components are integrated into the PMADE 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 AH. Furthermore, the PMADE and the additional system components are based on Java Virtual Machine to achieve a high portability and interoperability and to use standard interfaces to various operating systems.

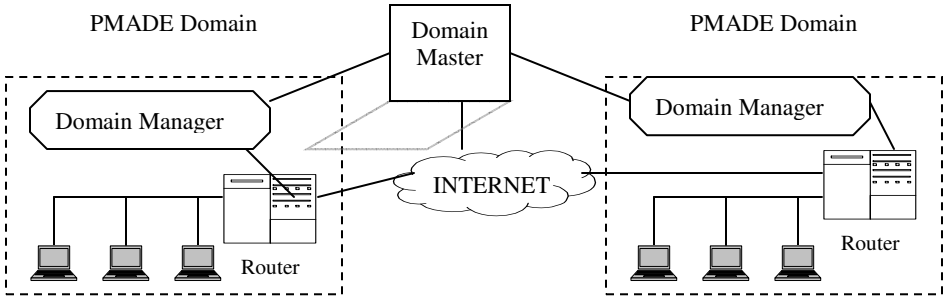

Fig. 2: PMADE domain concept

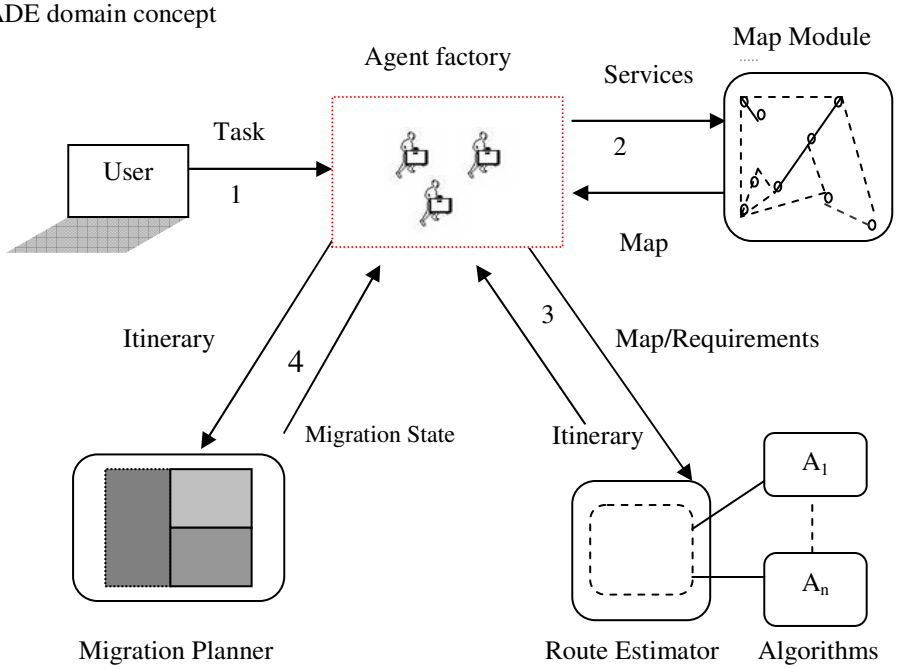

Fig. 3: An Agent logical view of the system

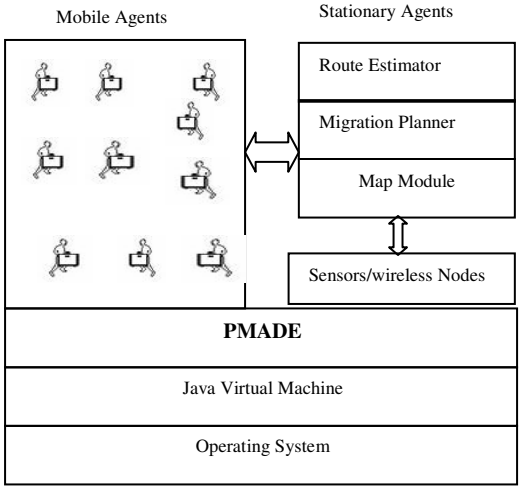

Fig. 4: System architecture

The system builds its network of AH on this principle. Many of these single AHs form the agent network infrastructure, which is structured into PMADE Domains. AH communicate only asynchronously/synchronously by transmitting mobile agents/messages, respectively.

Implementation: This section presents function and performance of the network latency and bandwidth monitoring sensors. Precision, actuality and reliability of the network measurements made by these sensors influence the Route estimating and migration planning directly. Implementation of the system is done on three laptops, 22 wireless computers and setup of Ethernet lab whereas one laptop acts as the Master as shown in Fig. 5. On this computer we launch a second AH which acts as a Master Domain Manager. On the other laptops we launch AH acting as another Domain Manager. All the Domain Managers run network sensors.

To understand the behavior of the sensors in different network situations, we constructed several test networks. The first scenario describes an inter-domain communication in between the same Ethernet link. All AHs and the associated network nodes are located in the same IP-subnet. Normally, two domains do not share such an environment. Hence, we use different communication ports for the Domain Managers.

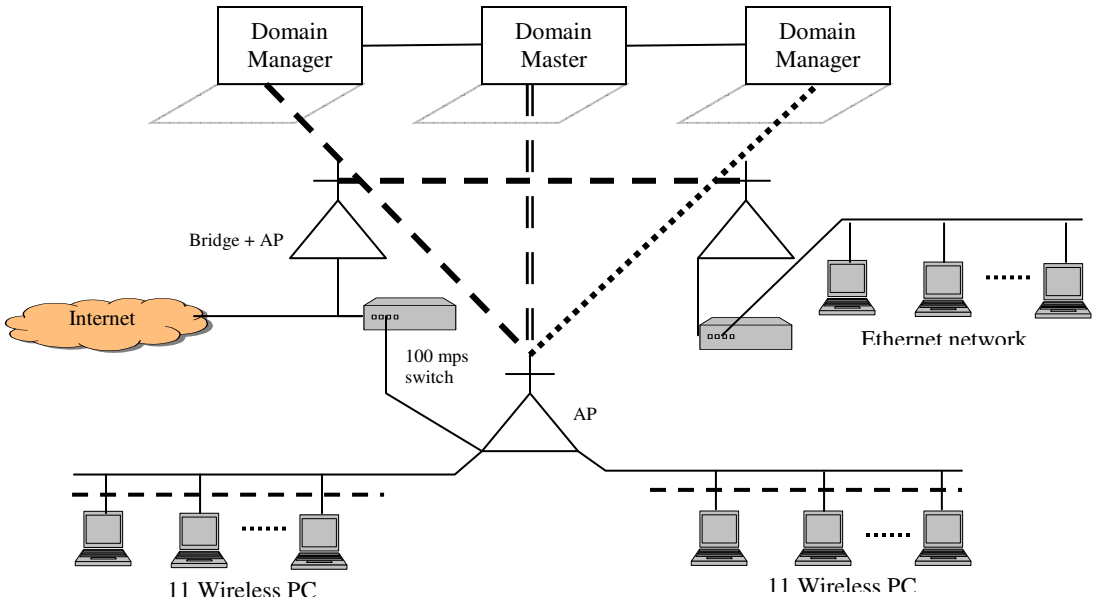

Fig. 5: Environment architecture

We use a 100 Mbps Ethernet with a 16-port switch. In the second scenario the remote domain/AH is located in another IP-subnet. The two 100 Mbps Ethernet based IP-subnets are separated by an I-router 915. The third scenario is like the first but one Domain Manager is a mobile node linked by three D-Link-2100 IEEE 802.11g WLAN access points. In the fourth scenario, an IEEE 802.11b W-LAN access point links all computers and in the fifth the mobile nodes communicate in ad-hoc mode. In all IEEE 802.11b cases we take care for the full 11 Mbps throughput of the wireless connection during the measurements.

MAP module: 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 Estimator and the Migration Planner to achieve optimizations.

Basically, a map of an AH consists of a partial network graph. The vertices of such a partial graph are the visible AH of the surrounding area such as all nodes in the local domain including the domain manager and the neighbored domain managers. 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 AH and a couple of network parameters that reflect the current performance of the connection. Since the PMADE comes as a Java application, we have naturally a lack of hard network information, because raw sockets are not supported in Java. There are possibilities to use basic operating system functionalities outside of Java (e.g. Ping) by using Java Native Interface (JNI[12]) e.g. Ping ICMP[13] for Windows systems. We try to avoid the use of JNI tools to preserve portability and interoperability as well as to avoid security problems. Thus, PMADE uses network sensors (note: we have used wireless machines as sensors in the testing of the proposed system) with interfering measurement methods on top of Java to get network information. The measurement environment and the sensors are described in the following subsections.

Ping gives back the round trip time of an ICMP packet and is a good indicator for the actual network traffic on the used link. Unfortunately, Java does not support raw sockets. Hence, the PMADE latency sensor emulates a Ping over a TCP connection. Thereby, an AH node opens a connection to a special port of the remote AH. After establishing the connection, it sends a small packet and starts the time measurement. The answer of the remote AH is an acknowledge, the measurement stops and the connection gets disconnected. We assume this procedure takes more time than the operation system Ping. Nevertheless, we use the OS-Ping as a reference to compare the performance of this sensor. During the measurements we found two main effects. PMADE latency sensor values are a little bit higher (about 1 to 2 ms) then the Ping values. We assume this offset comes from the TCP-controlled transmission of the sensor packet and the handing over times to Java. The second effect is that the deviations from average values increase according to the network load. Ping round trip times (RTT) deviations are significant smaller than the latency sensor RTT ones. This comes from the short Ping timeout causing Ping RTTs to be ignored, if longer delays occur. The latency sensor complies with the relative long TCP timeout and therefore, produces in this case values up to 10^4s.

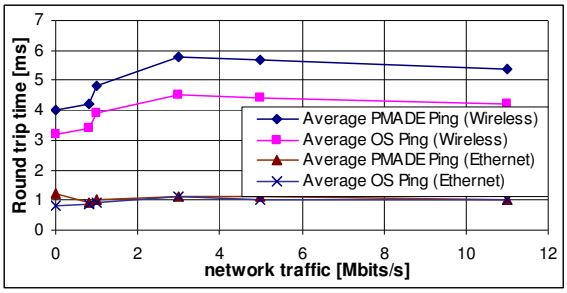

Fig. 6: RTTs measured by the PMADE latency sensor and by OS-Ping in different environments

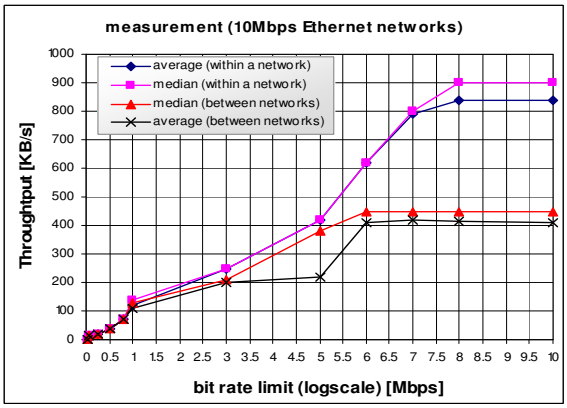

Fig. 7: Bandwidth inside an Ethernet and between two Ethernets (10 Mbps)

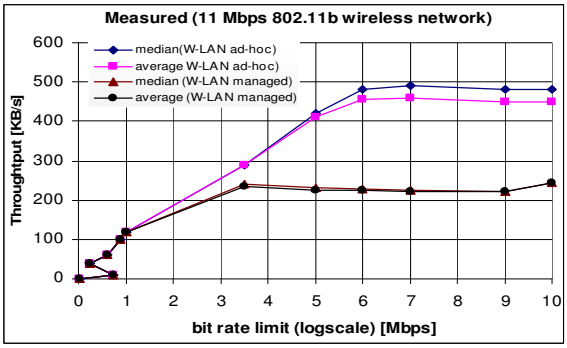

Fig. 8: Bandwidth in wireless LAN (managed and ad-hoc)

Figure 6 shows the average RTT measurements over 100 values corrected for deviations bigger than 2 * 10^1 ms. In spite of the nearly constant overhead to the Ping values, latency sensor values are comparable precise.

The bandwidth sensor works similar to the latency sensor but the sent packet is considerably larger. After finishing transmission and time measurement the bandwidth sensor calculates the current available bandwidth as the quotient of the given fixed packet size and the transmission time. Generally a big packet gives a more precise result but induces also a higher load on the network. Typically, the size of mobile agents (in PMADE) ranges from some hundred bytes to more than 5 Kbytes and increases as per application implementation. This is a rough orientation for a relevant packet size. But such small packets need only about 10 ms transmission time in a 10 Mbps Ethernet and the resolution of Java's time measurement is only 1 ms. After a preliminary examination with several packet sizes from 10 Kbyte to 500 Kbyte we decided to use a 30 Kbyte packet for our experiments which gives more reliable bandwidth results and loads the network rather minimal. To verify the bandwidth sensor's quality we limited the data output of one AH node with the Traffic Control tool and measured the available bandwidth to the remote AH node in several network scenarios. Figure 7 compares the measured bandwidth inside an Ethernet and between two Ethernets connected by a router. At a limited available bit rate of about 800 Kbps the throughput of the router-connected Ethernets falls off compared to the intra-Ethernet throughput. This effect results from more delays and collisions caused by the two collision domains and by delays in the router's forwarding process.

Figure 8 compares the bandwidth between two AH in a IEEE 802.11b wireless LAN in managed mode and ad-hoc mode. The double maximum bandwidth in ad-hoc mode comes from the direct communication. In managed mode all data goes over the access point to the other node, which means two data streams per time, on the shared radio media. The graphs turn into a horizontal, nearly straight line. At these points the natural throughput of the connections is reached. All measured bandwidth values correspond with the limited bit rate caused by the overhead of about 20%.

First tests have shown that the quality of measured data by PMADE sensors is high enough for route and migration optimizations. The extreme values of the latency measurements can be filtered out easily. We are going to make measurements with full duplex Ethernet to reduce collisions. In case of the Java time measurement granularity is improved, we hope to decrease the size of the packets used by the bandwidth sensor.

Route estimator: This module may be used by mobile agents to optimize the sequence of AH to visit, i.e., the itinerary. If an agent chooses a random path through a network, the sequence may lead to a non-optimal total migration time. The Route estimating process itself is basically the Traveling Salesman Problem (TSP)[14], 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[15]. The comparative performance of the algorithms depends on the problem and the given detailed circumstances.

The computation of an itinerary is based on the map data. We calculate a kind of distance matrix simply by using the reciprocal values of measured bandwidth. This matrix has to be updated at regular time intervals to fit the environment's dynamic behavior. Then, a pathfinder algorithm is applied in order to get a distance matrix with shortest paths between two places. In some experiments, we figured out that our distance matrix is not symmetrically in general. This is caused by variation in the bandwidth values and non-symmetrical connections measured by the Map Module. For TSP, there are algorithms for asymmetrical (ATSP)[16] and for symmetrical matrices (STSP)[17].

The variation in network throughput influences the result and success of the route estimating, especially short time variations. The Route Estimator generates an itinerary with a fast path through the net on basis of the distance matrix. Thereby, some of the best paths may be blocked by short-time traffic. At the point in time, when an agent uses the optimized itinerary, the generated path may not be the best one any more or, in the worst case, is by now the slowest one. The probability that this happens is lower in networks with clearly differing connection qualities. The Route estimating is especially useful in networks with different connection qualities and in networks with connections, which have different loads over a longer time period. In networks with nearly identical connection qualities, the use of Route estimating algorithms makes no sense – just choose a random path instead of spending time to calculate the random path.

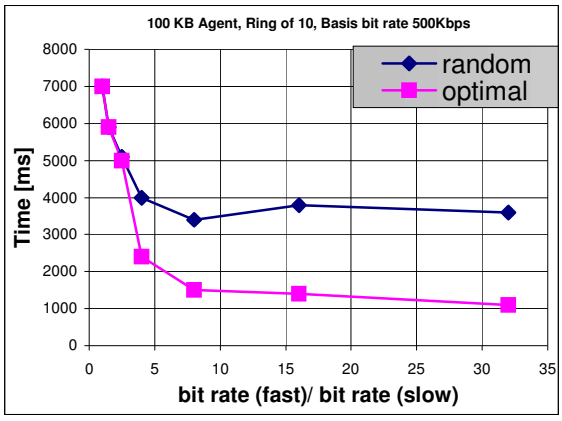

Fig. 9: Simple measurement experiment with 22 AH

We want to discuss the following scenario: a ring of 22 AH A1, A_2, A_3.... A_{22} connected with a 100 Mbps Ethernet and a bit rate limitation to 500 Kbps for one direction of the ring ($A_1 \rightarrow A_2 \rightarrow A_3 \rightarrow \ldots A_{10} \rightarrow A_{20} \rightarrow A_1$). We also have limited the other direction of the ring from 500 Kbps to 16 Mbps. We have measured the performance difference between the round trips of an agent which takes a random path and an agent which takes the optimal path. Figure 9 shows the measured the time for the round trips. The speed up of the optimal path agent increases, if the line speed gets better. Thus, Route estimating is useful.

Generally, our Route estimating process starts with a nearest neighbor search algorithm to generate a first path through the net. This path is input for further optimizations with an iterated 3-Opt algorithm (I3Opt). Figure 10 shows the result of the nearest neighbor algorithm which is about 36% above optimum (optimum means minimum in this case) but is calculated within 0.7 ms. This Route estimating is done on a generated matrix of the problem space triangulated random matrices (TMAT) with 100 places[18]. Such a matrix is an asymmetrical one where an entry is the shortest path between two places.

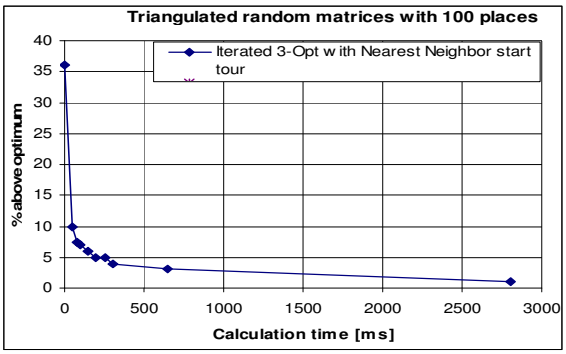

Fig. 10: Route estimating with TMAT 100

I3Opt is a local optimization algorithm which provides good solution in the implementation of the considered problem as shown Fig. 10. Steps evolved in the implementation are:

- Select 3 edges from the path
- Remove these edges (the path is broken into 3 parts)
- Insert these edges so that a better path is achieved
- Repeat until no improvements are achieved (local optimum)

The algorithm itself is also iterated several times to achieve an improvement. There is a good improvement after the first iterations and only moderate improvements for further iterations. I3Opt is a good algorithm to achieve a solution with only some iterations and a quite short computation time. The number of iterations can be chosen dynamically. We iterate this algorithm as long as the amount of migration time saved is not shorter than the computation time. Hence, the number of iterations depends on the computing power.

An initial start tour for I3Opt should be generated by a fast algorithm, which delivers a near optimal result. This is a trade off between computation time and quality of result. The nearest neighbor algorithm is very

fast but the calculated path is quite far away from optimum. An even better choice for ATSP is the patch algorithm. The algorithm chooses exactly one link from each vertex in a way that the sum of distances of all chosen edges is minimal. The result is a set of circles which have to be connected to one circle. For ATSP, patch is as good as nearest neighbor (and better). It reaches results about 10% above optimum.

Within the Route Estimator, we decided to integrate a combination of the patch and the I3Opt algorithms. It delivers good results for asymmetrical distance matrices. In a test run with tmat-instances, we reached a result within about 0.3 s which is 0.43% above optimum (100 AHs, P-IV 3.0 GHz, Java). In a case the distance matrix is symmetrical; the result of the patch algorithm is not as good as for the asymmetrical case. In spite of that, the I3Opt achieves good results for this case too. So this combination of patch and I3Opts seems to be a good choice.

Currently, we run a test sequence in a real network. Unfortunately, the number of AH within this network is small. There are 24 AHs. We have defined some random bit rate between 500 Kbps and 16 Mbps for every link. Connections are limited on the basis of such generated values to get a network with different connection qualities. Our combined algorithm reaches a solution that is about 3% above the optimum. Nearest neighbor reached about 13%. The migration times of the agents are reduced by about 50% compared to random paths. However, we have to do measurements in larger environments to make further qualitative statements.

Migration planner: During execution, an agent consists of three parts: the agent's state and data and its set of tasks (one or more task code to be executed at different nodes in the network- is set of class files). In Java, the state and data can be serialized for transmission. Such a serialized agent has to be transmitted to an AH to guarantee execution. Furthermore, an agent needs some portion of its tasks to be executed. The point of time when an agent's tasks are transmitted depends on the migration strategy – the way how a mobile agent is transmitted over the network. There are so called push strategies which transmit an agent's tasks along with the agent's state and data (before the agent is started at a remote AH). Using a pull strategy, an agent's tasks are downloaded dynamically (while the agent is executed at a remote AH) from its home site. The agent's home platform is the Agent Submitter (AS)[4] where the agent was started the first time. Furthermore, strategies can be distinguished by which tasks are transmitted: all tasks code at once or only some tasks. For example, the pull-all strategy means: transmit the serialized agent, start the agent at the remote site and in case that at least one task is required, download all tasks of the agent from its home. Using a push strategy, agent's tasks can be transmitted to the next AH of the agent's route or even to all AH visited by the agent. For example, the push-tasks-to-all strategy transmits first some of agent's tasks (those tasks which are needed potentially at remote AH) to all AH which are visited by the agent. Missed tasks will be downloaded dynamically. Then the serialized agent is migrated to the first AH of its itinerary. For the next hops, only the serialized agent is transmitted.

The Migration Planner is used to optimize time and network load caused by a transmission. Calculating the expected transmission times for different migration strategies does this. The results are compared to select a best fit migration strategy. In[4], a network model was developed for that purpose. It allows us to calculate network load and transmission time for migration of a mobile agent from home, between AH of its route and back home. For the computation, it takes in account an agent's size (state, data and tasks), data which is collected on its itinerary (increases with a constant factor) and connection qualities (latency and bandwidth). Thereby, a task is used at a remote AH with a certain probability.

In[4], this network model is also refined and extended. The data collected by an agent increases by a non-constant size and might be transmitted back home from an AH on the agent's route. Furthermore, code servers and mirror servers are added to the model. A code server is a server which contains all tasks of an agent. Such a server can be used by an agent to download tasks instead of downloading from home. A mirror server might be used by an agent to upload collected data instead transmitting data to the home site. An agent can initialize code and mirror servers on its route. With this extended network model, the effort and the advantage of initializing and using code and mirror servers can be computed.

There are some technical problems to determine the actual size of the serialized agent at runtime. For the comparison of different migration strategies, this size is constant and needs not to be involved in our computation. The same holds for the collected data. Hence, the Migration Planner compares the transmission time for the tasks of an agent. The number of tasks and the point in time of transmission differs for different strategies. Possible requests for task downloads have to be taken in account.

In more detail, a computation of the migration time for different migration strategies for a hop is done according to the following scheme: An agent wants to hop from server S_i to S_{i+1}. The agent's home server is S_0. The latency between two AH is defined by the function δ. Function τ denotes the available bandwidth between two AH. The amount of bytes which will be transmitted is B_c (size of all tasks) for

push-all-to-next is $T = \delta(S_i, S_{i+1}) + B_c/(\tau(S_i, S_{i+1}))$ and for pull-all is $T = \delta(S_0, S_{i+1}) + B_c/(\tau(S_0, S_{i+1}))$.

Furthermore, it is difficult to determine the probability for the usage of a certain task at a remote AH it is not designated if it is designated it can be very easily traced in PMADE. Thus, we decided to use the worst-case assumption that every task has to be downloaded as long as we do not have any other hints. A time computation can be made by pull-tasks

$$T = \sum_{k=1}^{n} \delta(S_0, S_{i+1}) + \left(B_c^k + B_\tau\right)/\tau(S_0, S_{i+1})$$

B_c^k is the size of the k-th task code of the agent. B_τ denotes the size of a request for downloading a certain task code.

In[8], we have developed some optimization variants. The basic process for an optimization is simple:

- Calculate migration times of different migration strategies
- Compare results and choose best migration strategy

In literature it is found that there is no overall optimal migration strategy. A mobile agent might use the Migration Planner to compute an optimal migration strategy regarding migration times for parts of its route or even for the whole route. A simple variant is to optimize the next hop only by comparing the migration strategies push-all-to-next, pull-all, pull-tasks and push-all-to-all. The algorithm looks like this:

```
/*Calculate transmission times*/
/*Push-all-to-next: Transmit tasks to next AH*/
T-patn = delay(S_i, S_j) +Task_size/bandwidth(S_i, S_j) ;
/*Pull-all: Download tasks from home site at next AH*/
T-pa = delay(S_0, S_j) +Task_size/bandwidth(S_0, S_j) ;
{Pull-task: Download each task from home at next AH}
T-pt = delay(S_0, S_j) + SUM (Probability( k )
*(Task( k )+Request))/ bandwidth(S_0, S_j) ;
/* Only for the first hop: push-all-to-all: and Distribute tasks
from home to all AH*/
for s in servers
{
    T-pata=T-pata+delay(S_0, S_s) +
    Task_size/bandwidth(S_0, S_s) ;
}
/*Select migration strategy*/
    T-min = T-patn;
    MS = "push-all-to-next";
if ( T-pa < T-min )
{
    T-min = T-pa;
    MS = "pull-all";
}Else if ( T-pt < T-min )
{
     T-min = T-pu;
    MS = "pull-tasks";
}
else ( T-pata < T-min )
{
    T-min = T-pata;
    MS = "push-all-to-all";
}
```

The migration strategy push-all-to-all can be used only at the home site. From there, all tasks are transmitted to all AH visited by the agent. Then, only the serialized agent needs to be transmitted between the AH of the itinerary. No additional tasks are necessary. A special case is also the last hop of a mobile agent. This is the migration back to the home AS. Thereby, the collected data and the serialized agent are transmitted only. Thus, there is no optimization for this hop. A similar optimization variant is to optimize the migration for more than one hop (not only for the next hop). The computation of transmission times is made for all migrations. Thereby, the migration strategy is fixed for all hops. This method can be improved, if the migration strategy is not fixed at all. The complexity of the computation is increased for this method. We have to check this method in more detail before we implement it.

An automatic code server initialization might be useful in a case where it takes more time to download tasks from the home AS than from a near code server with a fast connection. Such a code server is the code base for further migrations as long as there is a good connectivity. This is useful only for pull strategies (downloading tasks code dynamically). The optimization is simply based on a comparison of migration times with and without a code server initialization. With a low optimization degree, the module compares the migration time with the home AS as a code server and with a local code server on the current AH. A medium optimization degree is reached, if all available code servers are taken into account. As a variation of the low degree optimization, the migration times for further migrations with a dynamic code server initialization are computed (high optimization degree). The initialization of a mirror server depends on whether an agent wants to transmit collected data to home site. Collected data loads the network again and again when the agent migrates. In a case, where a mobile agent does not need this collected data for further computations, the data should be sent home site. Now, the Migration Planner computes whether it is cheaper to initialize a mirror server or to use the home AS to upload data. Automatic data upload variant calculates the migration time to the next AH, if all data is carried along with the agent. The result is compared with the time to upload collected data and to migrate without unnecessary data. The introduced optimization variants are some approaches to reduce network load and migration time of a mobile agent. New variants and combinations of variants result from the mentioned optimizations. The

implementation of these optimizations is not finished yet. Thus, we are not able to present measurement results at the moment. A complete evaluation of all optimization variants is also difficult due to the lack of a large global infrastructure for testing.

Analysis: 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 Planner to achieve optimizations. Map Module cares for precise and up-to-date knowledge (maps) within its local domain and provides a rough, summarized view of the linked remote domains. Map Module consists of several network sensors (Latency Sensors, Bandwidth Sensors) for computation. A mobile agent is able to locate points of interest within the network of AH (Map Module). Once a list of interesting AH has been determined, another system component - the Route Estimator can be used by the agent to plan an itinerary[8]. As long as we have an itinerary, a mobile agent may also use a so-called Migration Planner Module.

For computing the overhead on the network we have taken few assumptions which are- Total Nodes = T_N, Nodes of Interest = I_N. Packets per Node = N_p, Size of Packet = S_p, Requested Packet = R_p, u = Requested Packet (R_p)/Total Packets Available (N_p) and Traffic due to one node = $N_p . S_p$. Thus, Total Traffic due to all nodes of Interest can be given by

$$T_{I_N} = \sum_{1}^{I_P} N_P S_P$$

and Traffic due to relevant Packet on network will be $T_{I_P} = uS_P$. Overhead (O_{MM}) can be defined as (Total Traffic due to all nodes of Interest - Traffic due to Relevant Packet on network) is given by $\sum_{i=1}^{I_N} N_P S_P - uS_P \Rightarrow S_P \sum_{i=1}^{I_N} N_p - u$.

Route Estimator component is able to calculate the shortest trip through the net based on the map data. This module may be used by mobile agents to optimize the sequence of AH to visit, i.e., the itinerary. Itinerary is based on the map data. This component uses classic local optimization algorithms. Minimum path can be found out with the help of distance matrix. Distance matrix is calculated simply by using the reciprocal values of measured bandwidth and this matrix has to be updated dynamically. A path finder algorithm is applied in order to get a distance matrix with shortest paths between two places. We have assumed $\tau_i \rightarrow$ be measured bandwidth of i^{th} node and where i = 1 to N, the distance matrix D_i for N is given by

$$D_i = \begin{Bmatrix} a_{11} & a_{12} & ..... & a_{1N} \\ a_{21} & a_{22} & .... & a_{2N} \\ .... & .... & .... & .... \\ a_{N1} & a_{n2} & .... & a_{NN} \end{Bmatrix}$$

Thus, $a_{ii} = 1/\tau_{ii}$ where a_{ii} is calculated from the reciprocal value of τ_{ii}. Where a_{ii}'s stands for places or point of interest for which we find out the shortest distance. Assume P_i be a distance matrix that is found out from D_i by applying a pathfinder algorithm. Elements of P_i gives us the shortest path between two places and b_i's are the elements of matrix P_i. b_{ii}'s stands for shortest path between two places.

$$P_i = \begin{Bmatrix} b_{11} & b_{12} & .... & b_{1N} \\ b_{21} & b_{22} & .... & b_{2N} \\ .... & .... & .... & .... \\ b_{N1} & b_{N2} & .... & b_{NN} \end{Bmatrix}$$

Distance matrix is not symmetrically because of varying bandwidth values and non-symmetrical connections measured by the Map Module. For TSP, there are algorithms for asymmetrical (ATSP)[16] and for symmetrical matrices (STSP)[17]. As all of the working of Route Estimator is done locally so no traffic is present on the network to go from one place to another only algorithms are used for local calculations of path so we assume the overhead is negligible in this case.

During execution, an agent consists of three parts: the agent's state and data and its set of tasks. As long as we have an itinerary, a mobile agent may also use a Migration Planner Module to optimize each single migration included in the itinerary. 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 AH. This is also used to optimize time during transmission. Network load is the traffic on the network due the migration of mobile agents. Total traffic and overhead is calculated as given below so that we optimize the network load. Mobile agent carries its code and state across the network. For each hop j traffic is T_{MP}^j.

Traffic generated due migration planner is given by $T_{MP}^j = R_P C_{MA} S^j S_P$, where $R_P \rightarrow$ Requested Packet, $C_{MA} \rightarrow$ Code for mobile agent, $S^j \rightarrow$ Size of the state of agent at hope j and $S_P \rightarrow$ Size of a Packet. Thus, size of state of the agent is given by $S_j = d_{list} + \omega + \sum_{1}^{i} uS_P$, where $d_{list} \rightarrow$ Size of the list, $\omega \rightarrow$ Size of the other internal data structure

representing the state of computation $\sum_1^j uS_P \rightarrow$ Indicate the useful information collected by the agent at each, Visited node. d_{list} , u , S_P and ω do not depend on the node and for simplicity $\overline{\omega} = d_{list} + \omega$. Thus Overall Traffic $T_{MP} = \sum_{j=0}^{N}(R_P S_P + C_{MA} + \overline{\omega} + \sum_i^j uS_P)$ and Traffic Overhead are given by $O_{MP} = T_{MP} - 1$, i.e., $(R_P S_P + C_{MA} + \overline{\omega})(I_N + 1) + 1/2(I_N + 1) - 1$ $=> (R_P S_P + C_{MA} + \overline{\omega})(I_N + 1) + 1/2(I_N - 1)$.

RELATED WORK

In[4], authors refer to some performance evaluations[19,20] and comparisons[21,22] for mobile agent systems. These performance evaluations only consider one-hop migrations for non Java-based systems. The performance comparisons of different mobile agent systems turned out to be difficult. A comparison has to consider that each system has implemented different security strategies, different migration and transmission strategies, etc. Regarding migration times, a system with security issues is not comparable with a system without such a feature. Thus, we have compared PMADE with Aglets and other systems extensively. Furthermore as far as we know, there is no comparable framework for optimizing agent's travel through the agent systems infrastructure. Similar approaches to obtain network information can be found in the area of active networks. Such a network consists of active nodes. An active node is a network node which is able to detect smart packets and execute the embedded code e.g., a router with a special execution environment. In[23] active nodes on a path are used to measure characteristics of the path's sections as well as characteristics of the nodes. The measured characteristics are highly detailed. Our framework is able to detect intermediate network nodes and also a destination oriented end-to-end view. The information on the network characteristics is adequate for Route estimating and migration strategy decisions.

CONCLUSION AND FUTURE WORKS

In this paper we presented an evaluation of system components for a mobile agent system. These system components provide a framework for mobile agents to improve their performance as well as their autonomy and pro-activity for the navigation through the agent system framework infrastructure. For the evaluation, our focus was the quality of the modules as well as their performance advantage for mobile agents.

We took a look at some reference measurements in a network with the help of the OS-Ping. The comparison showed that the measured round trip time has a quite constant deviation and is nearly as good as the OS-Ping. Furthermore, we made some measurements in defined the quality of data which is collected by sensors of the Map Module. Therefore, we made network environments. The throughput measured by the bandwidth sensor reflects the actual network quality quite good. Caused by the bad granularity of time intervals in Java, we decided to do the bandwidth measurements with a quite large packet size.

We also made first measurements with the Route Estimator. Caused by the lack of a larger infrastructure, we had to use generated distance matrices to evaluate the performance of this module. In most cases, we assume the distance matrix is an asymmetrical one. Thus, we use a combination of algorithms for ATSP. The measurements showed that the module is able to calculate a nearly optimal path through the net within a very short time. We also started first measurements within a real network. Our first impression is that this Route estimating process works, but we are not able to present qualitative statements, as yet.

Finally, we introduced some optimization variants of the Migration Planner. Thereby, the computation of expected migration times is used for the selection of a migration strategy. This computation is based on the map's data. Variation in values falsifies the results also in this case. The comparison of migration strategies is based on the computation of the code tasks' transmission times.

With this framework, we made a qualitative step in the development of PMADE. This framework is an extension of PMADE, which is designed to be plug-able into other Java-based mobile agent systems. In the past, without this framework, the programmer or even the end user had to integrate routing information into PMADE agents. Now, agents can do this task on their own (in an autonomous fashion) and even more, dynamically adjust the itinerary. Thus, the performance gain cannot quantitatively be measured within our system.

The developed prototypes have to be updated and completed. Furthermore, a sample application will be developed to show the interaction of the framework's modules and to test their applicability in more detail. We are currently in the process of implementing this system for cluster computing and mobile transaction management.

REFERENCES

1. Picco, G.P., 2001. Mobile Agents: An Introduction. Microprocessors and Microsystems, 25: 65-74.
2. Tripathi, R., T. Ahmed and N.M. Karnik, 2001. Experiences and future challenges in mobile agents programming. Microprocessors and Microsystems, 25: 121-129.

3. Vigna, G., 1998. Mobile code technologies, paradigms and applications. Ph.D. Thesis. Politecnico di Milano, Italy.
4. Patel, R.B. and K. Garg, 2004. A new paradigm for mobile agent computing. WSEAS Trans. Computers, 1: 57-64.
5. Erfurth, C. and W. Rossak, 2002. Characterization and management of dynamical behavior in a system with mobile agents. In Innovative Internet Computing System-Second Intl. Workshop, IICS 2002, K¨uhlungsborn (Germany), Jun. LNCS 2346, H. Unger, T. B¨ohme and A. Mikler (Eds.), Springer-Verlag, pp: 109-119.
6. Patel, R.B., 2004. Design and Implementation of a secure mobile agent platform for distributed computing. Ph.D. Thesis. Department of Electronics and Computer Engineering, IIT Roorkee, India.
7. Patel, R.B., N. Mastorakis and K. Garg, 2005. Mobile agent location management in global networks. WSEAS Trans. Computers, 7: 697-710.
8. Patel, R.B. and K. Garg, 2005. A flexible security framework for mobile agent systems. Control and Intelligent Systems, 33: 175-183.
9. Schreiber, S., 2002. Beschreibung und Analyse von dynamischen Netzen f¨ur Agentensysteme. M. Sc. Thesis. Friedrich-Schiller-Universit¨at Jena, Institutf¨ur Informatik.
10. Wolski, R., N. Spring and C. Peterson, 1997. Implementing a performance forecasting system for meta-computing: The network weather service. Proc. Supercomputing'97 San Jose, CA: ACM SIGARCH and IEEE.
11. Fensch, C., 2001. Class splitting as a method to reduce network traffic in a mobile agent system. M. Sc. Thesis. Friedrich-Schiller-Universit¨at Jena, Institut f¨ur Informatik.
12. Java Native Interface, http://java.sun. com/j2se/1.3/docs/guide/jni/, 2003.
13. Ping functionality in java, http://www.geocities.com/SiliconValley/Bit/5716/ping/index eng.html, 2003.
14. Lin, S., 1965. Computer solutions of the traveling salesman problem. Bell System Technical J., 44: 2245–2269.
15. Johnson, D.S. and L.A. McGeoch, 1997. The traveling salesman problem: A case study in local optimization. Local Search in Combinatorial Optimization, E.H.L. Aarts and J.K. Lenstra (Eds.), John Wiley and Sons, Ltd., pp: 215–310.
16. Johnson, D.S., G. Gutin, L.A. McGeoch, A. Yeo, W. Zhang and A. Zverovitch, 2002. Experimental analysis of heuristics for the ATSP. The Traveling Salesman Problem and its Variations, G. Gutin and A. P. Punnen (Eds.) Kluwer Academic Publishers, pp: 445–487.
17. Johnson, D.S. and L.A. McGeoch, 2002. Experimental analysis of heuristics for the STSP. The Traveling Salesman Problem and its Variations, G. Gutin and A. P. Punnen, Eds. Kluwer Academic Publishers, pp: 369-444.
18. Cirasella, J., D.S. Johnson, L.A. McGeoch and W. Zhang, 2001. The asymmetric traveling salesman problem: Algorithms, instance generators and tests. Proc. 3rd ALENEX, 17 Jun. LNCS 2153, pp: 32-59, Springer-Verlag.
19. Gray, R.S., 1997. Agent Tcl: A flexible and secure mobile-agent system. Ph.D. Thesis. Department of Computer Science, Dartmouth College, University of Pennsylvania, Georgia, USA.
20. Johansen, D., N.P. Sudmann and R. van Renesse, 1997. Performance issues in TACOMA. Proc. 3rd ECOOP Workshop on mobile object systems: Operating system support for mobile object systems, Jyv¨alskyl¨a (Finland), C. Tschudin, J. Baumann and M. Shapiro (Eds).
21. Dikaiakos, M.D. and G. Samaras, 2000. Qualitative performance analysis of mobile agent systems: A hierarchical approach. Department of Computer Science, University of Cyprus, Tech. Rep..
22. Silva, L.M., G. Soares, P. Martins, V. Batista and L. Santos, 2000. Comparing the performance of mobile agent systems. J. Computer Communications, Special Issue on Mobile Software Agents for Telecommunications, 23: 769-778.
23. Li, Y. and L. Wolf, 2001. Collection of network information in active networks. ACM SIGOPS Operating Systems Rev., 35: 39-49.