Some Thoughts on Migration Intelligence for Mobile Agents

Christian Erfurth; Peter Braun; Wilhelm Rossak
Computer Science Department, Friedrich Schiller University Jena; 07740 Jena, Germany
Christian.Erfurth@informatik.uni-jena.de

Abstract

Mobile agents can be considered to be a new design paradigm in the area of distributed programming. This paper deals with problems of mobile agents which should be able to achieve a user-given task autonomously. In search of a "good enough" solution, the agent should be able to find new places and should move "fast enough" through the network. Can "migration intelligence" help to solve these problems?

1 Introduction

In the last years, research and development of mobile agents made a great leap forward. Along with the wide spread of Java based applications, mobile agents became extensively popular not only in research, but also in industrial projects. In the area of mobile agents, research looked at languages that are suitable for mobile agent programming (Knabe, 1995; Cugola et al., 1997) and languages for agent communication (Baumann et al., 1997). Very much effort was put into security issues (Vigna, 1998), control issues (Rothermel and Straßer, 1997; Baumann and Rothermel, 1998), and design issues (Hammer and Aerts, 1998). Several prototypes of real-world applications in the area of information retrieval, management of distributed systems, and mobile computing are in development (Pietro et al., 1999; Rothermel, 1997).

We look at mobile agents from the viewpoint of software engineering and distributed systems. They can be considered to be a new design paradigm in the area of distributed programming and a useful supplement of traditional techniques like the Client/Server architecture. As almost all other mobile agent research groups, we also have a rather pragmatic notion of the term mobile agent. To our understanding, it is any kind of software entity that is able to initiate a migration on its own within a network of heterogeneous computer systems. In addition, it works autonomously and communicates with other agents and host systems.

At the University of Jena we focus on research on all aspects related to migration which are important for performance aspects. One of our main ideas is that mobile agents must be able to influence the migration process to be able to adapt to changing requirements, e. g. network parameters. To support our research, none of the existing prototypes was useful, because in these systems the process of code migration is not open to the programmer. In most existing systems, Java code migration is implemented simply using standard Java techniques. Therefore, we developed our own mobile agent system, named TRACY Braun et al. (2001a, 2000), in which we can change and configure almost all migration related aspects. TRACY is a general-purpose mobile agent system, i. e. it serves as a foundation of both research and application development in the field of mobile agents.

In this paper we want to present a classification of migration aspects and present problems in conjunction with migration optimization. To solve migration optimization problems, we divided our viewpoint into two levels: a micro and a macro level. Concepts for possible solutions will be discussed and compared. At last, an intelligent migration strategy classificator is proposed as a first step towards a possible implementation.

2 Migration Aspects and Open Problems

In a network node, a mobile agent resides in a special mobile agent server which is its execution environment. The migration process interrupts the agent's execution and the mobile agent is packed and sent over a network connection to another mobile agent server to resume execution. Usually the mobile agent initiates the migration process by itself. During the self-initiated migration, the agent carries all its code and the complete execution state with it.

We can define three aspects of the migration process and distinguish between different kinds of migration (Braun et al., 2000). First, from the *programmer's point of view* the migration process can be classified as weak or strong mobility (Braun, 1999). Secondly, from the *agent's point of view* the mobile agent can choose from different migration strategies, i. e. how migration is done (pull code, push code, etc.). Finally, the *network's point of view* is how code and data is transmitted (protocol level), called the transmission strategy.

Strong mobility makes it possible to interrupt the mo-

bile agent's execution (for migration) and to resume the execution (after migration) at the *next statement* within the agent's code. In case of weak mobility, the execution cannot be resumed at the next statement but a certain (possibly predefined) *method* of the agent is invoked after migration. This method is the entry point after migration.

The mobile agent also has the choice to migrate using a certain strategy. It can combine different ways to ship code, data and state over the network to meet specific needs (see section 2.1). This is the agent's independent decision; therefore, we call it the agent's point of view.

For actual data transmission a suitable protocol has to be used. For that purpose a new protocol can be designed or existing protocols like TCP/IP or UDP can be used. A protocol can also be combined with code compression. This is the lowest level of the migration process and is called the network's point of view.

For research on these migration related topics our especially developed mobile agent system TRACY offers weak mobility (programmer's point of view), several migration strategies (agent's point of view) and different transmission strategies (network's point of view) (Braun et al., 2000). For this paper the agent's point of view is the focus because an optimized migration strategy can possibly be chosen autonomously by the agent with the help of "intelligence".

2.1 Migration Strategies

A method to ship mobile code over the network is called a migration strategy. Not only code has to be shipped but also the execution state of the mobile agent and data the agent carries along. There are various migration strategies. One possibility is to send the complete *program* (whole code) over the network. The opposite is to transmit only certain required parts (units) of the code. Another choice is whether to push code over the network, i. e. code will be sent over the network in advance, or to pull (download) code from a reachable location, i. e. the mobile agent (the execution unit) loads code from some suitable source. If the push code variant is used, code could be sent to the next location only, or to all locations on the agent's itinerary.

So a migration strategy can be classified by answering the following questions (figure 1):

1. How much code is transmitted?
   (a) complete code which belongs to the agent
   (b) those parts of the code which are potentially needed at a remote platform
2. When is code transmitted?
   (a) pull code after migrating to a remote platform
   (b) push code before migrating to a remote platform
3. Where to is it transmitted (only push code)?
   (a) to one, i. e. the next location
   (b) to all locations the agent will visit

However, regardless of the chosen strategy, the execution state of the mobile agent and the data must be transmitted. In figure 1 we present a graphical version of this classification. "Whole" code means the program code of the agent with all referenced classes.

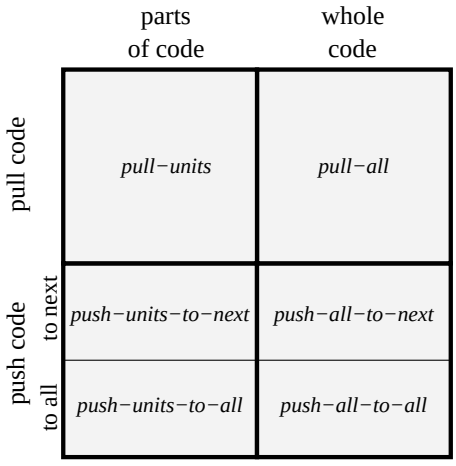

Figure 1: Classification of Migration Strategies

In case of the push-units-to-all and -to-next strategies, the units needed at remote platform have to be determined in advance. Otherwise, absent units must be dynamically loaded using the pull-units strategy.

2.2 Migration Optimization

There are several issues which can effect the agents execution and the optimization of migration. For instance, if the push-all-to-next strategy (push the complete code to the next platform) is used, the agent carries along code which is possibly never used. However, this makes sense in the case that the original code platform (providing the agent's code and all related classes) is not reachable from each location on the agent's itinerary. If the agent chooses the pull-units strategy and the parts that should be dynamically downloaded are quite small, then the communication overhead may become too expensive. However, the pull-units strategy might be a good choice if the complete code is only needed at a few agent servers on the itinerary. So, the network load is quite low and transmission times are shorter thereby.

Our optimization goals are to cause minimal network load and minimal migration times not only to the next location of the agent's itinerary, but also for the whole itinerary. This task is difficult to achieve because the itinerary has to be known in advance and, for optimization, the conditions at the itinerary stations have to be known in advance too.

A short example is shown in figure 2, where the agent makes a round trip. Connections between adjacent locations on the itinerary have a good connection quality, but direct connections back to the start platform are quite bad. Which migration strategy should be chosen? The decision can be explained using figure 1. For this scenario, pulling code directly from the start platform is expensive regarding time to transmit code; no matter whether the whole code or only parts are transmitted. So we could discard pull strategies. The same as to the pull strategies applies to the push-to-all strategies, because code has to be sent to all nodes in advance, using the slow network connections. It seems to be the best to take the code along on the trip using the push-all-to-next strategy. This should help to get lower migration times between the locations and a better migration time for the whole trip. The network load is possibly higher for this push strategy than for a pull strategy in case the complete code is not needed.

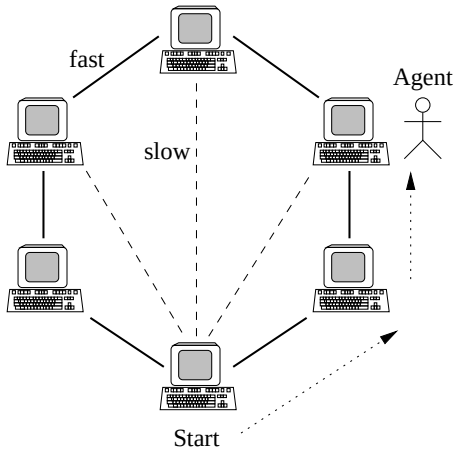

Figure 2: Agent's round trip in a special constellation

Partly there are complex correlations between different constellations, network situations, agent's characteristics, and the situation at the locations. If the itinerary of the agent is known in advance, and the situation at the locations could be estimated roughly, network load and transmission time could be pre-estimated, or even precisely calculated for mobile agents with special characteristics. But is this enough?

The main goals are to get an optimal solution for a user-defined task, to use an optimal migration itinerary for this task, to cause minimum network load and to use a minimum of time for migration (whole trip). These goals can be divided into two classes:

- Macro level: optimal solution of a given task
  - the given task should be fulfilled by the agent
  - the solution should be optimal for the agent's owner
  - the itinerary is optimized with regard to user-level problem solving
- Micro level: to solve migration problems in an optimal manner
  - the agent should work with optimal strategies
  - network load and timing are optimized
  - the itinerary is optimized with regard to physical network characteristics

2.2.1 Micro Level

The *micro level* is defined to deal with questions which are connected to migration, especially the strategy of migration and transmission. As mentioned above, there are various migration strategies from which the agent can choose. The decision which strategy is chosen for the migration to the next location is made within this level. The agent can choose the strategy just for the migration to the next location or for the whole trip.

On the one hand, a strategy has to be found which leads to minimal load and minimal transmission time with respect to the network. On the other hand, with respect to the mobile agent, a migration strategy has to be chosen which leads to a minimal load and migration time for the whole trip. The network load the agent causes is the amount of data which has to be transmitted over the network. It is the sum of the complete network traffic caused by the agent. If the goal is to optimize the migration to the next location, the caused traffic to this location is measured; if the goal is to optimize the whole trip of the agent, the complete necessary traffic is measured. Migration time is the time which is needed to migrate, i. e. is needed to transmit the amount of data. It is the sum of the migration times needed to transmit code parts, again like above to the next location or to all locations on the itinerary. For the decision regarding a suitable migration strategy, the agent needs to know about the itinerary, i. e. about agent servers which should be visited, network qualities, and so on.

On the micro level, we made some performance measurements for various migration strategies. Thereby, some parameters (see section 4) could be determined which can be used to choose an optimal migration strategy. It can be shown that there is no single optimal migration strategy (Braun et al., 2001b). The determination of the parameter values can be done by program analysis during agent execution. There are migration strategies where the agent needs statistics or rules of thumb to choose the correct migration strategy. Collecting stats and drawing conclusions, the agent could *learn* to move "fast enough" through the network.

2.2.2 Macro Level

On the *macro level*, we focus on questions which are connected to the solution of user-given tasks. This level is on top of the micro level; migration aspects are not considered within this level.

The optimization goal within this level is to find a user-optimal solution for a given task. This implies for the agent to visit suitable locations or to communicate and cooperate with other agents. Suitable locations are locations where the agent can find suitable services regarding the asked-for solution.

Figure 3 shows a hierarchy for task solution. The mobile agent acts on behalf of a user. This means the user has to hand over a task to one or more mobile agents. Such a mobile agent communicates and cooperates with other agents and has to use the services available in the network, provided by the agent servers. To contact other agents or to find suitable services, a migration might be necessary. The migration process is part of the micro level.

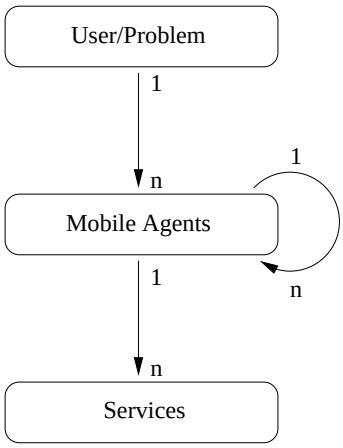

Figure 3: Task Solution Hierarchy

The macro level deals with problems like finding mobile agent servers and services, and with optimizations for the itinerary (possibly based on an itinerary given by the user). At each agent server, there has to be information (or hints) available that allow to discover and to locate other suitable agent servers (in order to fulfill the task). Consequently, the mobile agent has to *learn* to move through a network which might include agent servers unknown to the agent.

2.3 Levels, Data and Intelligence

The different goals of the Macro and the Micro Level result in different data requirements. We have to dicuss which data are needed for each optimization level and how to aquire data (level-and-data problem). Furthermore, we have to check whether intelligence helps to analyze data (data-and-intelligence problem). Lastly, we have to discuss where intelligence can be used within the levels and what kind of intelligence can be used (level-and-intelligence problem) (see figure 4).

Within the Micro Level, the optimization process needs more detailed data: information on the itinerary, regarding the agent and the network are needed. Information on the itinerary is provided by macro level optimization. Agent code analyzes supply information on agent characteristics. A physical view of the network is needed to provide information on network characteristics (see figure 5), like band width, connection state, etc. The different line styles used in the figure characterize different network characteristics. The X-line means a brocken connection.

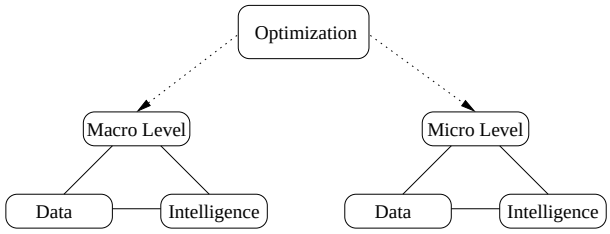

Figure 4: Optimization Levels

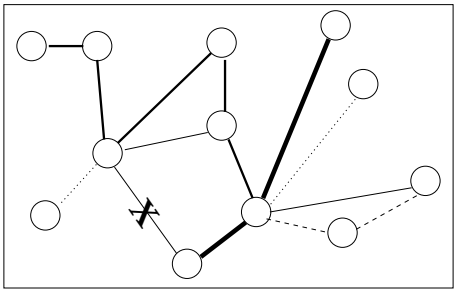

Figure 5: Physical View of the Network

In contrast to the Micro Level, it are the network nodes which are interesting for Macro Level optimization, especially the services provided by the agent servers. So a logical view of the network holds the needed information on services and agent servers (see figure 6). Different node colors within the network shown in the figure indicate different services.

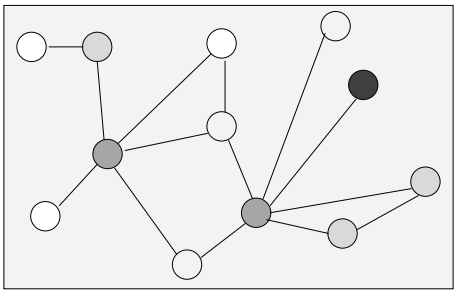

Figure 6: Logical View of the Network

The descriptions of the physical and logical network are input for optimizations. After the Macro Level optimization, an optimized itinerary is the result. This itinerary and the description of the physical network are

input for the Micro Level optimization which leads to a suitable migration strategy chosen for the agent's trip. Possibly the itinerary, as found on the macro level, has to be changed because of bad or broken connections which are only seen on the micro level (see figure 7).

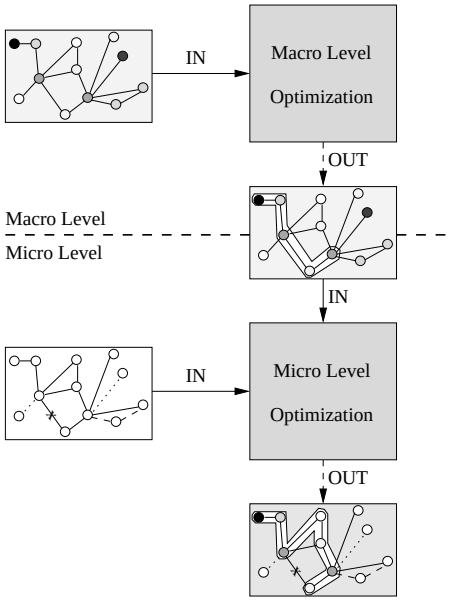

Figure 7: Optimization Levels

3 Possible Solutions

We want to solve the optimization problems within both levels. The solution we want to achieve for the optimization problem within the micro level is to get minimal network load and migration times. Within the macro level, we want to reach a certain user-specific quality.

On the micro level, we need to know the next or even all locations on the itinerary, network qualities, and agent characteristics. On the macro level, we need to know where to find suitable services, helpful hints, or new agent servers. Thus, on both levels, there has to be information available for mobile agents to solve the optimization problems. So we have to solve the level-and-data problem noted above (see figure 4). However, if we discuss possible solutions, we can choose two principal different approaches – a central solution and a distributed solution.

3.1 A Central Solution

In the *central solution* approach, a global database has to hold all information on mobile agent servers, services provided by these servers, and information on network qualities.

The basic problem of this approach is that there may be a lot of data to store, even for only a few agent servers (if that data is available at all in a dynamically changing environment). Furthermore, every agent has to connect to the central server for fulfilling its task and every agent server has to register at the central server and provide information or network connection status to the central server. This contradicts the principles of the otherwise fully distributed agent architecture and may become a bottleneck.

3.2 A Distributed Solution

Using the *distributed solution* approach means that the information on servers, services and network qualities has to be distributed. Each single mobile agent server has to provide information about the available services and the network quality.

We can imagine that every mobile agent (server) has a "map" providing this information (independent from the chosen approach). These maps may differ in terms of scope, size and the provided level of detail. A discussion of the different possibilities follows now.

3.2.1 A Global Network Map

A first map we can imagine is a *global network map* which provides all information about the complete network of agent servers. With such a map, we run into the same problems as with the central solution. Moreover, problems with data acquisition are inherent in a large network. The highly redundant data has to be held up to date. In addition, a global network map introduces again a centralized concept in an otherwise distributed solution. Figure 8 shows a global network map. The viewpoint (node where agent resides) is the black colored node. The agent can "see" the whole network.

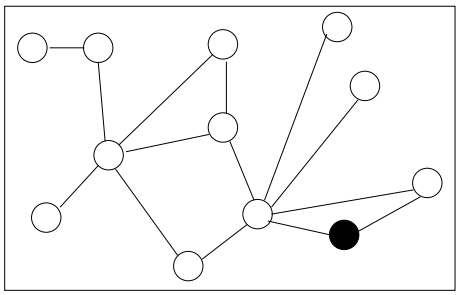

Figure 8: Global Map

However, if it is of no concern that information may become obsolete for further away regions, this concept could make sense.

3.2.2 A Neighbor Map

The second type of map regarded here is a *neighbor map*. Such a map describes the view from one agent server to the directly adjacent agent servers. There may be different

variants to define adjacent servers, e. g. all servers within a subnetwork and, between subnetworks, two (user-) defined servers are adjacent, or all servers reachable within a certain time are adjacent. This is not the scope of the paper. In any case, only the direct neighbors are visible, only information on the next neighbors is stored. There is no information on more distant agent servers and the mobile agent has to migrate to find further hints for a solution (like a physical search: move to find). A neighbor map view is shown in figure 9 – only the white marked area is visible to the agent.

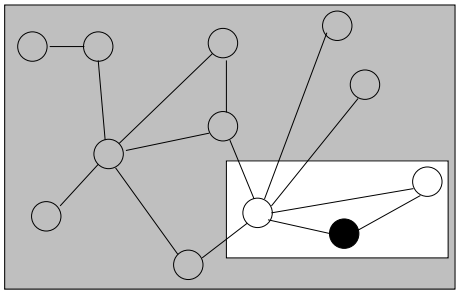

Figure 9: Neighbor Map

The obvious advantage is, that the information which has to be collected by the agent server is reduced to a minimum.

3.2.3 A Map of the Surrounding Area

A last and more interesting type of map is a *map of the surrounding area*. This map reduces information – the further away, the more blurred (fish-eye view). All information on the local region is stored. Information is reduced more and more for agent servers, services, and network qualities which are not within the local region. The reduction is made during data acquisition. The white marked area in figure 10 is sharp – all information is available to the agent. This area could be larger than the neighbor area. Reduced/blurred information is available on the light grey marked area. Minimal or even no information is known for the dark grey area. There can be more areas than noted here in this example.

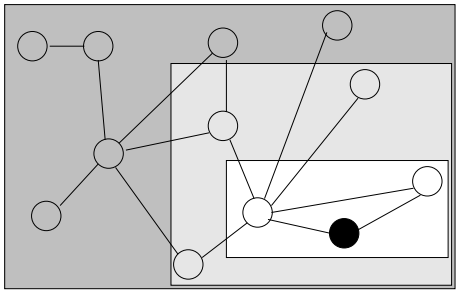

Figure 10: Map of the Surrounding Area

In this approach, the amount of data stored at an agent server can be reduced by focusing on relevant information. The map provided by an agent server is the local viewpoint of this agent server. Only information from the local region has to be collected.

With the help of such a map, the agent can move through the network using fully structured data or only hints (blurred information; approximated data) depending how far away the target is. New services and unknown agent servers can be found by using hints. The mobile agent has to *learn* to use hints and to *learn* to move in a "good enough" manner.

If we look closer at this approach, the global network map and the neighbor map are special cases of the map of the surrounding area.

3.3 Comparison

This comparison is a discussion of the data-and-intelligence problem (see figure 4). The completeness of information (data) is compared with the need for intelligence.

In the central solution, the mobile agent needs only a minimum of intelligence to find mobile agent servers and services or to choose the right strategy for migration. "Intelligence" is needed to build, represent, and update a good database containing network and agent system status and to provide best answers using an information system. This also applies to the global map of the distributed solution. However, a large amount of data has to be held at every agent server. This may cause an enormous communication overhead, especially if the agent server network may be dynamical.

However, perfect migration optimization is only possible, if all information regarding the itinerary and the agent are known at the start of migration. Therefore, only the central approach or the distributed approach with a global map can be used to get a proven global optimum.

If we use the concept of a "map of the surrounding area" (distributed solution), the mobile agent needs rules of thumb to learn to move through the network and to learn to choose the right strategy for migration based on an incomplete set of information. How much "intelligence" is needed by the agent depends on the level of information which is available to the agent.

This means thats we "trade" *intelligence* vs. *information*. The less information is available, the more the need for intelligence. The lack of information must be made up by intelligence. If there is no information, the level of intelligence must be theoretically infinite. Otherwise, if we can access the complete information, we can calculate and optimize the itinerary in a pretty straight forward manner. The result is a *perfect* solution. However, we also have to remember that some optimization problems are NP-complete and can only be approximated.

In the case of the minimal "neighbor map" approach, the mobile agent has little chance to make well founded

migration decisions, with or without intelligence. The information provided is simply not sufficient.

Concluding, it can be said that the *completeness of information* is most important for the solution. Thus, the difference between a central or a distributed solution is not the major problem; the provided information may be equal for the central solution and the distributed solution with a global map.

However, we have to look carefully at the completeness of information and the related need of intelligence. Figure 11 summarizes the difficulty of the solution and typical problems.

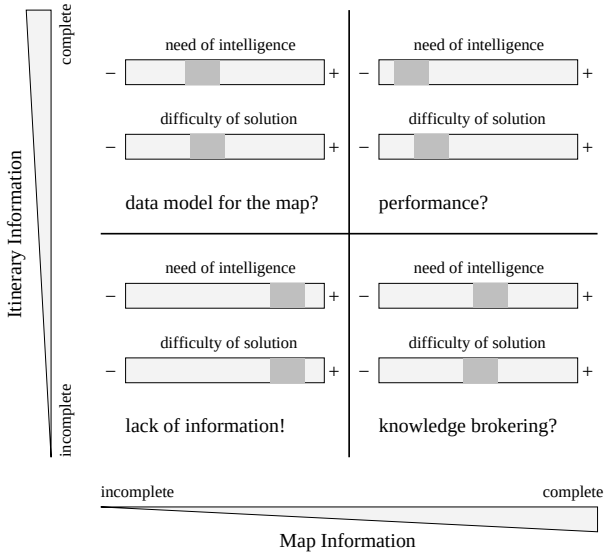

Figure 11: Level of Information Completeness and Related Need for Intelligence

If there is complete information, intelligence could be low and the difficulty of the solution is quite low (right upper quadrant). A problem is the performance for the calculation of a perfect solution. In contrast, if the information provided by a map is reduced (left upper quadrant), the need for intelligence is higher. The probability to find a solution can be increased furthermore with a higher level of intelligence. The question is how much the information can be reduced without effecting the chance to find a good solution. There must be a good data model available. At the right lower quadrant, the itinerary is not completely known in advance, but the map is complete. This situation is more complex than the one mentioned before. Even more intelligence is needed to solve the task. The problem here is adequate knowledge brokering. In the last quadrant in the lower left, a lot of intelligence is needed to fulfill a task. There is a basic lack of information and, therefore, the probability to find a solution is very low. By moving through the network of agent servers the mobile agent can aquire information on the network – rising the chance to find a solution despite of almost impossible odds to make intelligent decisions.

Regarding the data-and-intelligence problem it can be said that the more blurred the data is, the higher is the need for intelligence. If the computation for an optimized solution is not too hard, intelligent mechanisms do not make sense (in the case that all information is available). If the map has reduced information and the itinerary is known, we could mainly do an optimization within the micro level, with the help of intelligence. However, with a reduced map the itinerary might be partly unknown. If the map is not reduced and the itinerary is not known completely, we could use intelligence to optimize mainly within the macro level. If both types of information are reduced we have to optimized at the micro and macro level (with reduced chances). The necessary level of intelligence is possibly to high, caused by the lack of data. For our research, two cases are interesting: either only map information is reduced or only information on the itinerary is reduced (right lower and left upper quadrants in figure 11).

Regarding the level-and-intelligence problem (see figure 4), there are two kinds of intelligence. The intelligence needed within the Macro Level is *Routing Intelligence* (right lower quadrant in figure 11). In this case, only information on the itinerary is incomplete. With the help of intelligence an itinerary for the mobile agent has to be built. In the case, only the map information is incomplete, the mobile agent has to choose a suitable migration strategy by using intelligence. This kind of intelligence within the Micro Level is *Migration Intelligence* (left upper quadrant in figure 11). We can improve figure 4 to a more concrete variant (see figure 12).

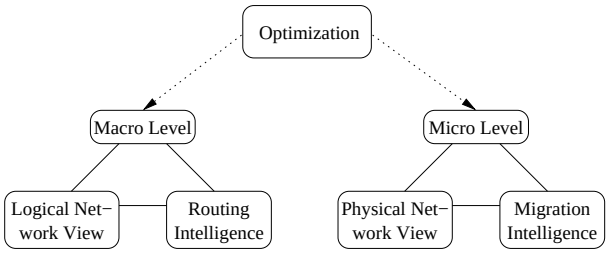

Figure 12: Optimization Levels

4 An Intelligent Migration Strategy Classificator

As a first step we want to build a classificator to find a suitable migration strategy. A *suitable* migration strategy means a strategy which meets requirements like low migration times or low network load. This classificator is a classificator located within the micro level. It could be used in case the itinerary is known and a map with reduced information is used (see figure 11).

In some cases, the decision to use a concrete strategy for migration can be based on a calculation. For that, the

itinerary, the agent characteristics, and the possibility for usage of a certain code part at a remote location have to be known in advance. With the performance model designed in Braun et al. (2001b), the network load and the transmission time can be calculated.

From the software engineering point of view, we would like to regard the classificator as a black box with interchangeable content functionality which gets various input parameters and delivers a suggestion for suitable migration strategies (see figure 13). The classificator should supply a weighting for various migration strategies. At last, because of the autonomy of a mobile agent, the mobile agent decides itself which migration strategy is used. One realization of the black box would be a mathematical calculation, as considered above. Another realization would be an "intelligent" approach, e. g. neural networks or rule-based systems, combined with reduced map information. In our future work, we want to compare different approaches based on a minimum set of input parameters.

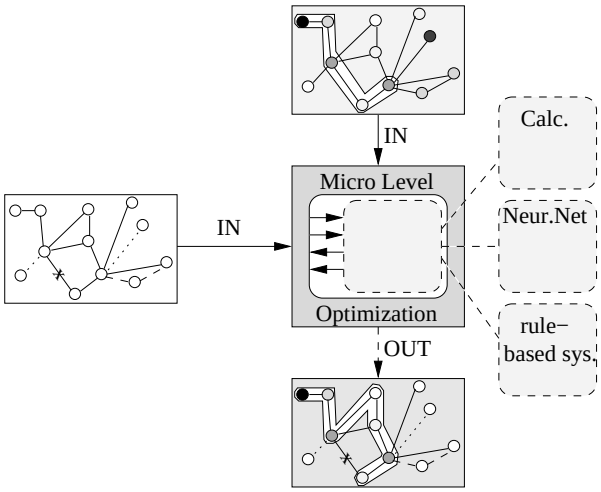

Figure 13: Classificator with interchangeable intelligent module

As a result of our performance measurements, noted above, we are able to pinpoint some parameters which universally effect migration time and network load:

1. Parameters concerning the agent
   - code size of units (e. g. classes)
   - size of data
2. Parameters concerning network
   (a) quantitative
      - band width
      - latency
   (b) qualitative
      - reliability

As a first approach for the intelligent module we use evolving neural networks (Pasemann, 1998). Before usage, these neural networks need to evolve themselves. At the beginning of the evolution there are only input and output neurons and no connections between. The evolution process is done by simulation. With the help of a suitable "fitting function", the neural network begins to create an internal structure to meet the requirements (fitting function). Normally, these networks are not pure feed forward networks, but include recurrent connections (feedback connections).

The problem is to define a suitable fitting function to get good results. A suitable fitness function may be the time for migration needed by the agent with its chosen migration strategy, or if we fix the time, the distance the agent covers. During the evolution of the neural network, the classificator needs feedback. This is the learning phase of the neural network where a validation of the chosen migration strategy is made.

The classificator itself could be located either within the mobile agent system or within the individual agent. On the one hand, integrated within the mobile agent system, the classificator has to be designed to support diverse mobile agents' characteristics. Thus, the number and type of input parameters may be varying for different agent characteristics. On the other hand, if it is integrated within the agent, the classificator can be designed agent-specific. But there is additional code and data to carry along increasing the agent's size. We decided to integrate the classificator within the agent because in this case the parameters can be fixed for each agent type. However, every agent server needs mechanisms to collect information from the surrounding network area and to provide it to mobile agents.

5 Conclusion and Future Work

We can achieve a truly optimized migration path only by using the central solution or the global network map within a distributed solution. These solutions are, however, not feasible for a larger set of mobile agent servers. We need a compromise between enough information and necessary intelligence to be "good enough". Therefore, the mobile agent needs *Migration Intelligence* within the Micro Level, i. e. intelligence that should help the agent to make autonomous decisions in terms of migration strategies, and *Routing Intelligence* within the Macro Level. The better the information, the less the need for intelligence (the more blurred the information, the higher the need for intelligence). An optimal compromise could thus be found using the concept of a "map of the surrounding area".

In our future work we plan to further verify the discussed solution. We will determine suitable parameters to provide "good enough" information for mobile agent tasks and will have to define a description language for

the logical and physical view of the network. To put "intelligence" into the mobile agent (server) we want to verify the use of evolving neural networks (Pasemann, 1998). The overall effort to put intelligence into the mobile agent (server) has to be checked.

Acknowledgments

We want to thank Prof. Frank Pasemann (TheoLab, Ernst-Haeckel-Haus, University of Jena), who provided us with the concept of evolving neural networks and triggered our investigations into intelligence for mobile agents. We thank also Astrid Lubinski (University of Rostock) for her input regarding fish-eye concept.

References

Joachim Baumann, Fritz Hohl, Nikolaos Radouniklis, Kurt Rothermel, and Markus Straßer. Communication concepts for mobile agent systems. In Rothermel (1997), pages 123–135. URL `http://www.informatik.uni-stuttgart.de/ipvr/vs/projekte/mole.html`.

Joachim Baumann and Kurt Rothermel. The shadow approach: An orphan detection protocol for mobile agents. Technical Report 1998/08, Universität Stuttgart, Fakultät für Informatik, 1998. URL `http://www.informatik.uni-stuttgart.de/ipvr/vs/projekte/mole.html`.

Peter Braun. Über die Migration bei mobilen Agenten. Technical Report Math/Inf/99/13, Friedrich-Schiller-Universität Jena, Institut für Informatik, 1999.

Peter Braun, Jan Eismann, Christian Erfurth, and Wilhelm Rossak. Tracy – A Prototype of an Architected Middleware to Support Mobile Agents. In *Proceedings of the 8th Annual IEEE Conference and Workshop on the Engineering of Computer Based Systems (ECBS), Washington D.C. (USA), April 2001*, 2001a.

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

Peter Braun, Christian Erfurth, and Wilhelm Rossak. Performance Evaluation of Various Migration Strategies for Mobile Agents. In *Fachtagung Kommunikation in verteilten Systemen (KiVS 2001), Hamburg (Germany), February*, 2001b.

Gianpaolo Cugola, Carlo Ghezzi, Gian Pietro Picco, and Giovanni Vigna. Analyzing mobile code languages. In Jan Vitek and Christian Tschudin, editors, *Mobile Object Systems: Towards the Programmable Internet (MOS'96), Linz, July 1996*, volume 1222 of *Lecture Notes in Computer Science*, pages 93–110, Berlin, 1997. Springer Verlag. URL `http://www.polito.it/~picco/papers/-ecoop96.ps.gz`.

Dieter K. Hammer and Ad T. M. Aerts. Mobile Agent Architectures: What are the Design Issues? In *Proceedings International Conference and Workshop on Engineering of Computer-Based Systems (ECBS'98), Jerusalem, March/April 1998*, pages 272–280. IEEE Computer Society Press, 1998.

Frederick C. Knabe. *Language Support for Mobile Agents*. PhD thesis, Carnegie Mellon University, Paittsburgh, Pa., December 1995.

Frank Pasemann. Evolving neurocontrollers for balancing an inverted pendulum. *Network: Computation in Neural Systems*, pages 495–511, 1998.

Eleonora Di Pietro, Oranzio Tomarchio, Giancarlo Iannizzotto, and Massimo Villari. Experiences in the use of Mobile Agents for developing distributed applications. In *Workshop su Sistemi Distribuiti: Algoritmi, Architetture e Linguaggi (WSDAAL'99), L'Aquila (Italy), September 1999*, 1999.

Kurt Rothermel, editor. *Proceedings of the First International Workshop on Mobile Agents (MA'97), Berlin, April 1997*, volume 1219 of *Lecture Notes in Computer Science*, Berlin, 1997. Springer Verlag.

Kurt Rothermel and Markus Straßer. A protocol for preserving the exactly-once property of mobile agents. Technical Report 1997/18, Universität Stuttgart, Fakultät für Informatik, 1997.

Giovanni Vigna. *Mobile Agents and Securtiy*, volume 1419 of *Lecture Notes in Computer Science*. Springer Verlag, New York, 1998. ISBN 3-540-64792-9 (paperback).