University of Salzburg
Institute of Computer Science

SatMAS
(JESICA)

Input pattern selection in a Multi Agent System

Thomas Aschauer
Thorsten Engl

Date: May 17, 2001
Version: 1.0

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter gives an overview about motivation for input pattern selection in an MAS. Additionally we give an overview of already existing methods of resolution.

1.1 Introduction to Artificial Neural Networks

1.1.1 Introduction to Artificial Intelligence

Science arises from the very human desire to understand and control the world. The invention of electronic computers greatly enhanced the human ability to model the scientific and technical problems into programs, and to solve them in an astonishingly fast way. Traditionally, problems were encoded into programs in the following steps:

1. Scientists research and understand a scientific model
2. Reduce this model into a few formula or rules
3. Code these formulas and rules into computer readable languages

Then after the program was developed, it could be used with different input data to solve a particular kind of problem. In analyzing this problem-solving model, we could figure out two presumptions: first, people should have full knowledge of the steps to solve the problem; second, these steps could be encoded into computer understandable languages. And in this traditional model, programs and computers had no intelligence of their own; they only conducted what programmers told them to do. But as people's ability to understand the world increased, they found the two presumptions were hardly true for many complicated problems, which were too complex to be reduced into a serial of predefined steps, and too difficult to program by hand.

As biological science was highly developed, people found out that each biological unit was just a physical unit that followed a set of very simple rules, but it could solve a large amount of complex problems, even problems they had never met before. So why not let electronic computers mimic those biological units, and while programmed to follow a set of simple pre-programmed rules, to solve complex problem in an innovative way. In this sense, let computers became "intelligent". So inspired by the biological evolution, there emerged the new technology of artificial intelligence.

1.1.2 What is an ANN?

An ANN[1] is an information-processing system that is based on generalizations of human cognition or neural biology.
The key features of NN consists of (taken from [1]):

- Information processing occurs at many simple elements called neurons.
- Signals are passed between neurons over connection links.
- Each connection link has an associated weight, which, in a typical neural net, multiplies the signal transmitted.
- Each neuron applies an activation function (usually nonlinear) to its net input (sum of weighted input signals) to determine its output signal.

A neural network (NN) is characterized by its particular:

- Architecture; its pattern of connections between the neurons.
- Learning Algorithm; its method of determining the weights on the connections.
- Activation function; which determines its output.

1.2 Introduction to Multi Agent Systems

MAS[2] are computational systems in which two or more agents interact or work together to perform some set of tasks or to satisfy some set of goals. These systems may be comprised of homogeneous or heterogeneous agents. An agent in the system is considered a locus of problem-solving activity, it operates asynchronously with respect to other agents, and it has a certain level of autonomy. Agent autonomy relates to an agents ability to make its own decisions about what activities to do, when to do them, what type of information should be communicated and to whom, and how to assimilate the information received.
Characteristics of Agents:

Autonomy An agent works independent from his host.

Single minded The agent performs a clear predefined job, his behavior is corresponded to this task.

Reactive The agents program run is event driven.

Environment dependent Interfaces to his environment and available resources determine his activities.

Permanent State information stay for the whole program run.

[1]Acronym for Artificial Neural Network. Short form: NN
[2]Acronym for Multi Agent System

Additionally, Agents may have some optional characteristics:

Interactive Communication between different agents is possible.

Mobile The agent moves between different hosts.

Adaptive He automatically adjusts himself to changed boundary conditions.

1.2.1 Mixture of Experts

A mixture of experts is a probabilistic model that can be interpreted as a mixture model for estimating conditional probability distributions. The model consists of a gating network that divides the problem into smaller problems and makes expert networks specialize on each of these sub problems. In terms of a mixture model the expert networks correspond to conditional component densities and a gating network to input dependent mixture coefficients.
Note that, the gating network splits the data in a "soft" way, allowing several experts to be selected at a time. Since the gating networks deals with the decomposition in smaller tasks the choice of the type of gating network is an important one.
There a three well known types for gating networks:

- Single Layer Perceptron with a soft-max activation function (standard mixture of experts model)
- Multi Layer Perceptron with a soft-max output activation function (also known as gated experts)
- Using Gaussian kernels to divi de the input space with soft hyper-ellipsoids

For complex compositions sometimes a hierarchical mixture of experts is used. This mixture has a tree structure, where the leaves contain the expert networks and the non-terminal nodes contain the gating networks.

A short overview of these gating network types is given in [2]

1.2.2 Cooperation in MAS

One of the key problems in cooperative MAS is how to get agents to cooperate effectively [3]. The need to interact in such systems occurs because agents solve sub-problems that are interdependent, either through contention for resources or through relationships among the sub-problems. These relationships arise from two basic situations related to the natural decomposition of domain problem solving into sub problems.

- The first situation is where the subproblems are the same or overlapping, but different agents have either alternative methods or data that can be used to generate a solution.
  For example, in a distributed situation assessment application, overlapping subproblems occur when different agents are interpreting data from different sensors (independent information sources) that have overlapping sensor regions (cover similar information).

- Another form of interdependence occurs when two sub-problems are part of a larger problem in which a solution to the larger problem requires that certain constraints exist among the solutions to its subproblems.
  For example, in a distributed expert system application involving the design of an artifact where each agent is responsible for the design of a different component (subproblem), there are constraints among these subproblems that must be adhered to if the individual component designs will mesh together into an acceptable overall design.
  Depending upon the character of subproblem interdependencies, the interactions among agents in a MAS can be complex, often requiring a multistep dialogue similar to an asynchronous co-routine type of inter-action.

Knowledge Query and Manipulation Language

KQML [3] is a standardised language for inter agent communication. In KQML messages are called Performatives. The syntax of these Performatives follows the Common Lisp Polish Prefix Notation.

Blackboard Architecture

The blackboard architecture is a design pattern that supports systems where nondeterministic solving strategies are used [4].

In many cases there are no known strategies how agents bring their results together. This is where the blackboard architecture takes place. There will be a "blackboard", data storage, where all elements of the solution space and corresponding control information are hold. Each agent sends its output to the blackboard and a central control component decides if the agents solution is plausible. Agents have the ability to use already existing solutions from other agents to establish a new hypothesis. The control component may reject an existing hypothesis or declare it as the final solution.

1.3 Motivation for Input Pattern Selection

When training an ANN we often face the problem of huge amounts of possible training data. This would result in extreme time consuming training cycles if all available data is used for teaching the network. So it's important to select only the essential part of the training pattern and keep the training data set as small as possible.

On the other hand, minimization of generalization error has also to be guaranteed, in order to ensure that the trained network can properly yield an optimal result. The selection of training data presented to the neural network influences whether or not the network learns a particular task. Like a child, how well a network will learn depends on the examples presented. A good set of examples, which illustrate the tasks to be learned well, is necessary for the desired learning to take place. The set of training examples must also reflect the variability in the patterns that the network will encounter after training. At first glance this appears to be a contradictory pair of objectives. However, just as the generalization error must be as low as possible, data sampling which involves

[3] Acronym for Knowledge Query and Manipulation Language

both collection and measurement of data is expensive and therefore needs to be reduced to a minimum. There are several methods of resolution for selecting a well fitting subset of the original (in most cases very huge) data set. However, the problem of selecting the optimal training set has not yet been solved.

1.4 Existing Methods of Input Pattern Selection

- Active Learning
  - Active Selection
  - Active Sampling
- Dynamic Pattern Selection
- Training Data Selection with Genetic Algorithms
- Training Data Selection with Multi Layer Neural Networks

1.4.1 Active Selection

The starting point for active learning is the observation that the traditional approach of randomly selecting training samples leads to large, highly redundant training sets. Such training sets can be obtained if the learner is enabled to select those training data that he/she expects to be most informative. In this case, the learner is no longer a passive recipient of information but takes an active role in the selection of the training data.
A deeper introduction to active learning can be achieved in [5].

1.4.2 Active Sampling

Recent research has shown active learning methods to be effective in increasing the modeling reliability of a neural network system. An active learning agent has the ability to query its environment in order to make a selection of its training data. One approach to the implementation of active leaning is to use querying-by-committee. This results in considerably reduced data collection and at the same time does not compromise the accuracy of identification. A nonlinear plant with both clean and noisy data is successfully modeled by such a technique and a feed forward neural network controller based upon such a model is demonstrated to perform effectively.

Minimized Data Collection - Active Querying Example

This is a data gathering method based on active querying. In this method data is reduced to a minimum, yet modeling accuracy is not compromised. The active querying criterion is determined by whether or not several neural network models agree when they are fitted to random sub samples of a small amount of collected data.
For details see [6].

1.4.3 Dynamic Pattern Selection

In contrast to active pattern selection, the dynamic pattern selection algorithm achieves concise training sets by continually validating the generalization properties of the net. Details of this method can be found in [7].

1.4.4 Training Data Selection with Genetic Algorithms

In this method a genetic algorithm is employed for the parallel selection of appropriate input pattern for the training data set.
For an example see [8].

1.4.5 Training Data Selection with Multi Layer Neural Networks

This method selects a small number of training data, which guarantee both generalization and fast training of the MLNNs applied to pattern classification. The generalization will be satisfied using the data locate close to the boundary if the pattern classes. However, if these data are only used in the training, convergence is slow. Therefore the MLNN is first trained using some number of the data, which are randomly selected (Step 1). The data, for which the output error is relatively large, are selected. Furthermore, they are paired with the nearest data belong to the different class. The newly selected data are further paired with the nearest data. Finally, pairs of data, which locate close to the boundary, can be found. Using these pairs of the data, the MLNNs are further trained(Step 2). Since, there are some variations to combine Steps 1 and 2, the proposed method can be applied to both off-line and on-line training. The proposed method can reduce the number of the training data, at the same time, can hasten the training.
A detailed description can be found at [9].

Chapter 2

Project specification

2.1 Satellite Image Classification

Since mankind is able to take pictures from outer space, it has always been a difficult task to recognize specific patterns, related to a special problem. As computer science raised a stadium where computers are able to perform some ”intelligent” tasks, a wide research area established in solving the problem of automatic image classification. There have been many different methods of resolution, ranging from genetic algorithms to neural networks. One common problems in all these attempts is that applications of biological methods are extreme time intensive.
Nowadays distributed systems are well known and popular used, so it's the logical consequence applying this technology to the problem domain of image classification. One possible way achieving a collaboration of these technologies is by using a multi agent system.

2.1.1 Data Structure

One important characteristic of satellite images is the wavelength used for sampling. Each wavelength range is distinguished by determined advantages and disadvantages, which places it best in certain application areas. Table 2.1 lists the most common used wavelength ranges, which was taken from [10].

Table 2.1: Wavelengths in the electromagnetic spectrum routinely used for various remote sensing applications

Wavelength range	Notation	Application related areas
0.40-0.50 μm	blue	Water penetration and water depth
0.50-0.60 μm	green	Vegetation greenness, ocean colour
0.60-0.70 μm	red	Chlorophyll absorption in healthy plants, iron oxide content in soils, water sediment load
0.70-0.90 μm	near IR	Healthy vegetation response, crop monitoring and classification, land and water separation; vegetation and soil separation; separation of built and vegetated surface cover
1.55-1.75 μm	near IR	Soil moisture content
2.00-2.40 μm	SWIR	Presence of clay-based minerals
3.00-4.00 μm	mid IR + thermal IR	Volcanic activity, bush fires, underground fires
9.00-12.50 μm	far IR + thermal IR	Earth ocean and land temperatures
2.4-3.75 cm	X-band microwave	Forest canopy shape, crop classification. Ocean roughness,wind speed
3.75-7.5 cm	C-band microwave	Crown thickness, leaf/branch size and orientation. Plant morphology. Ocean surface roughness, wind speed; oil seeps; surface elevation; land cover; bathymetry; geology; gravity fields; sea-ice and iceberg monitoring
15-30 cm	L-band microwave	Trunk size and tree density; ocean roughness; soil surface roughness; soil moisture content; forest clearcut areas
30-100 cm	P-band microwave	Trunk size and tree density; soil moisture content; soil surface penetration and under surface phenomena; snow penetration

2.1.2 Sample Images

In our problem domain the used wavelengths are the X- and the P-Band. Pictures 2.1 and 2.2 represent the same geographical area, where picture 2.1 was sampled in the X-Band and picture 2.2 in the P-Band.

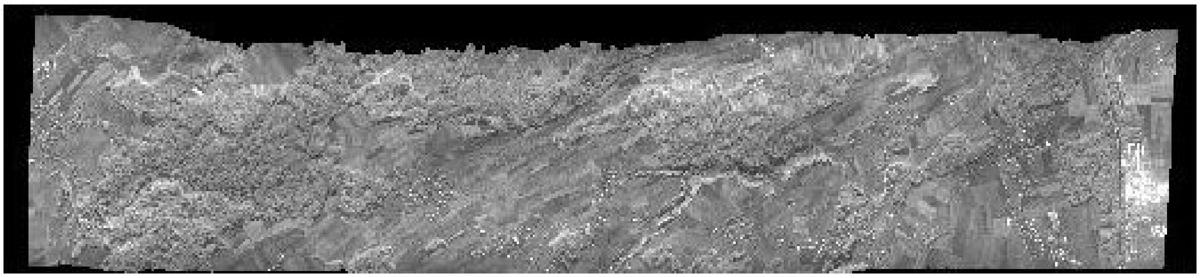

Figure 2.1: X-Band example

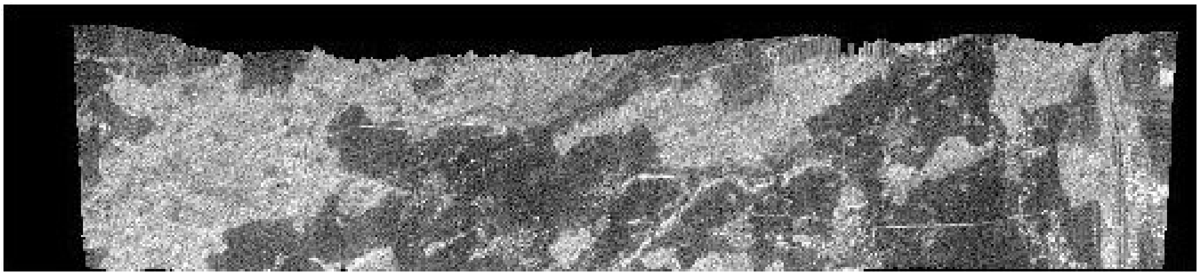

Figure 2.2: P-Band example

2.1.3 Example for a Classification

Find a properly visualization of such classifications during research is always difficult and time consuming. Therefore using some powerful tools is a duty. In our case YARB [1], an ultimate visualization system, fitted best.
In picture 2.3 you can see a classification of such a satellite image printed with YARB.

[1] **Y**et **A**nother **R**esult **B**rowser

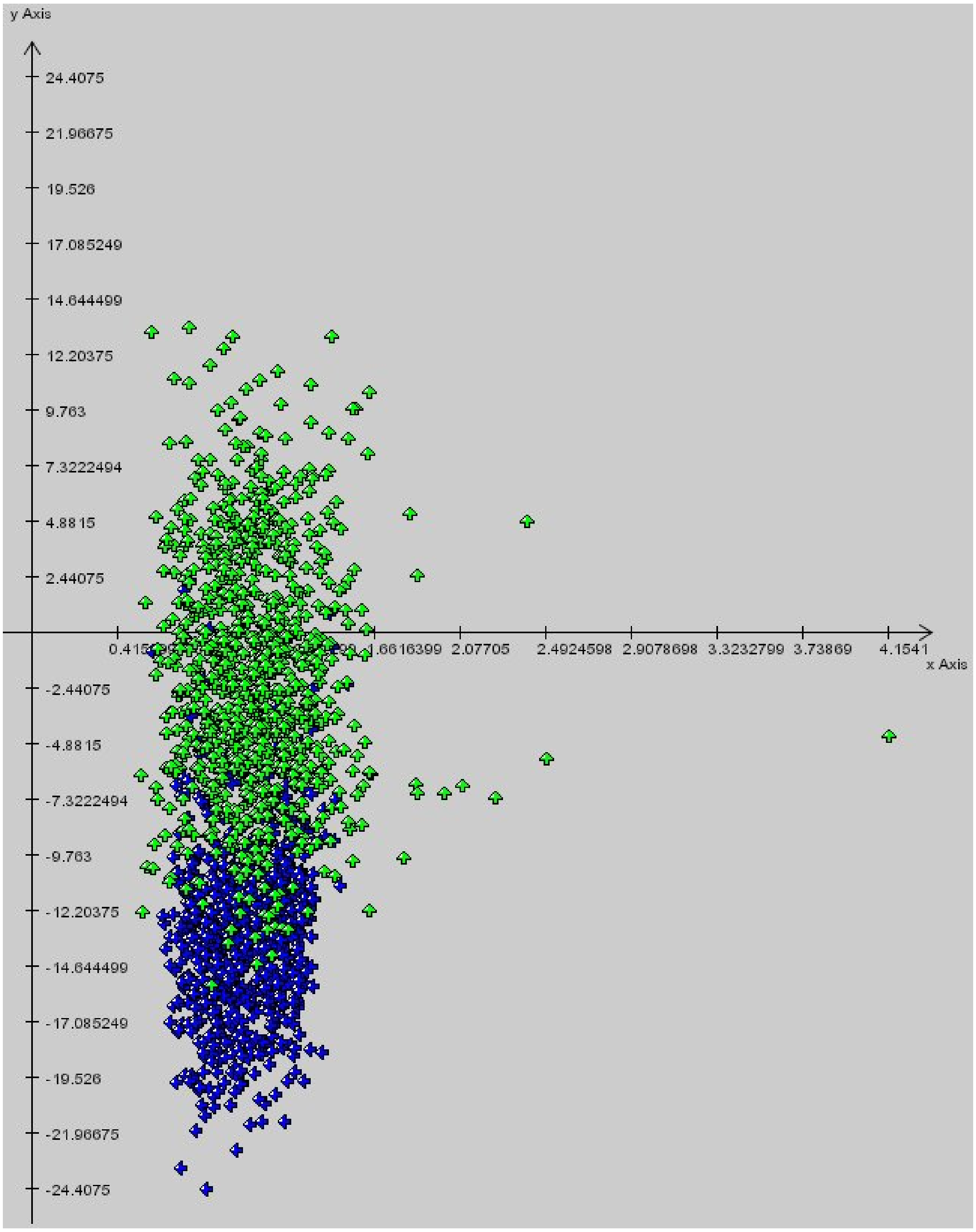

Figure 2.3: Sat-Example (Viewed with YARB)

2.2 System Architecture

There are many boundary conditions to be thought of, and we will discuss every part of them in a separate section:

- Multiple agents
- Local experts, ability to make requests for reinforcement
- Autonomous movement of agents
- Controller
- Distributed system, communication
- World modeling

Agents

The system should be built of different agents. Each agent has its own operational area and should have a high degree of autonomy. It is free to move around its area and collect data samples, which is used to learn the associated classification neural network. But there is not only a NN for classification, the movement control of the agent is also realized by a neural network.
The gathered data samples are hold in a "bag pack", that has a limited capacity. The decision which data samples are valuable is made by the agent itself.

The expertise of an expert should be convergent to the maximum, but when the data distribution of its area is too wide spread, this can't be guaranteed. In this case the expert has the ability to ask for reinforcement so that his area will be split into a distinct number of sub areas and each area will be inspected by a own agent. To prevent the loss of the old agents gained experience a nice extension would be to implement mechanism for transferring knowledge between agents. There are already research approaches on such mechanisms, like the one in [11].

Controller

After the learning period of all the agents is finished, the user may ask the controller for the classification results. The responsibility of the controller is then to ask the agents, which are experts of a distinct area, and bring together the corresponding results.

Distribution and communication

One boundary condition is to decrease the time consumption needed for full classify a satellite image. This is done by distributing the agents to multiple computer systems. Therefore a stable and efficient communication system is needed. As the preferred implementation language is Java, the method of choice will be RMI.

World modeling

Using different frequency bands leads to a big problem for system modeling: Should the agents move on a discrete area filled with data samples or on a net based on the real measure points? The advantage of the former method is that agent movement is trivial and it is easy to determine the experts of a specified area. The problem would be that there wont be a data sample on each point of the square grid, and even large areas of the grid may have almost no samples. As illustration of this problem take a look at picture 2.3, where the mass of data samples are only in a relatively small zone.

Another approach would be to create a net consisting only of the sample points. But then it would be difficult to calculate the movement of an agent and also find the best experts of an area.

Chapter 3

Software Design

Literature:

- General software analysis and design: [12, pages 741-822]
- UML: [13]
- Software pattern (Design pattern): [14], [15], [4]

3.1 General design aims

When designing our software to meet the requirements as described in the section for system architecture (Section 2.2), we had to take care of several boundary conditions:

- Flexible implementation of agent movement control
- Use a NN-library but in a replaceable way
- Easy change of world representation
- Concentrate the agent creation process in a central module
- Information interchange between different system components should be based on calls and callbacks (avoid polling over process boundaries)
- Multi threaded agent training
- User interface should be separated from the agent controller
- Agent state extracted from implementation to reduce network traffic

3.1.1 Naming conventions

As in every project where more than one developer is working on, SatMAS needs naming conventions, too. We designed our style guide to be not too strength (because most programmers don't like to be told how to do their work). Although naming should be done by general good software engineering practices, the most common naming conventions are given here.

Interface Names

In the SatMAS design all interface classes have the prefix *I*. Example: *IWorld*.

Property methods

In addition to the classes provided by the JDK all property methods have set-, get-, is-, or update prefix followed by the name of the controlled property. (e. g. getName or setParent).

Variables

Our variable naming convention is nearly the same as the famous *hungarian notation* in the Windows programming world. There, all variables have a lower prefix with their corresponding type, which can be one of the following:

integer: Prefix: *n* (e. g. nIndex)

float: Prefix: *f* (e. g. fAverage)

String: Prefix: *str* (e. g. strFilePath)

boolean: Prefix: *b* (e. g. bResult)

If the type is none of these, you should give the variable a really speaking name. For member variable it is mandatory to put the prefix m_ to it (e. g. m_nArrayCounter).
NOTE: This naming conventions are not mandatory for loop variables.

3.2 Component Diagramms

Components of the system and their responsibilities are shown here:

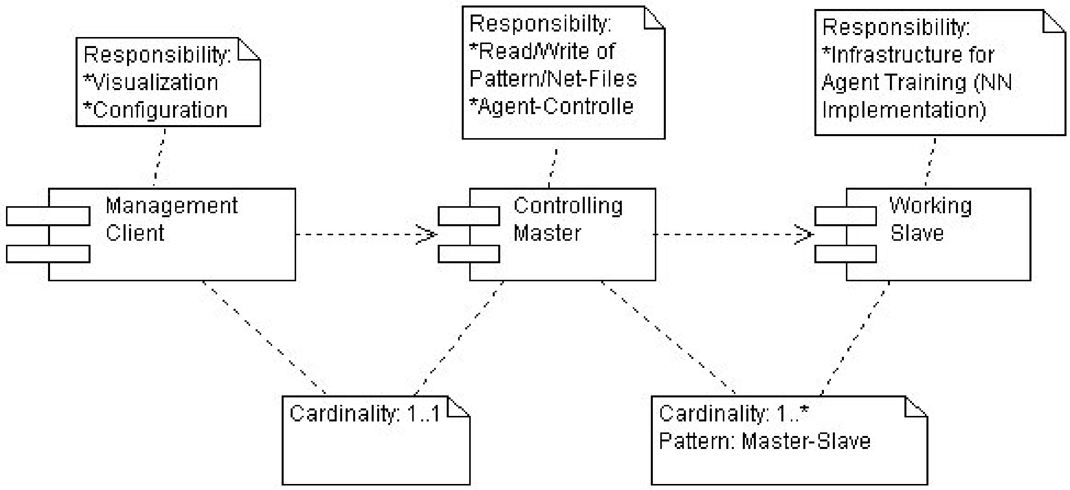

Figure 3.1: Component diagram: System Components

3.2.1 Management Client

The Management Client is the user interface of SatMAS.

Responsibility

- Visualization
- Configuration

Collaborators

- Controlling Master

3.2.2 Controlling Master

The Controlling Master is SatMAS's central intelligence component. Cooperation between Controlling Master and Working Slave is implemented as Master-Slave design pattern as described in [4, pages 245-291].

Responsibility

- Agent controller
- Disposition of agents
- Query results
- Creation of agents
- Input/Output of pattern/net files
- Notify client

Collaborators

- Management Client
- Working Slave

3.2.3 Working Slave

All time consuming operations of SatMAS are processed on a Working Slave.

Responsibility

- Provide infrastructure for agent training
- Notify master after training of an agent has finished

Collaborators

- Controlling Master

3.3 Class diagrams

3.3.1 Agent model

In order to give the Controlling Master the ability to build agents with different behaviors, the agents are modeled as seen in figure 3.2.

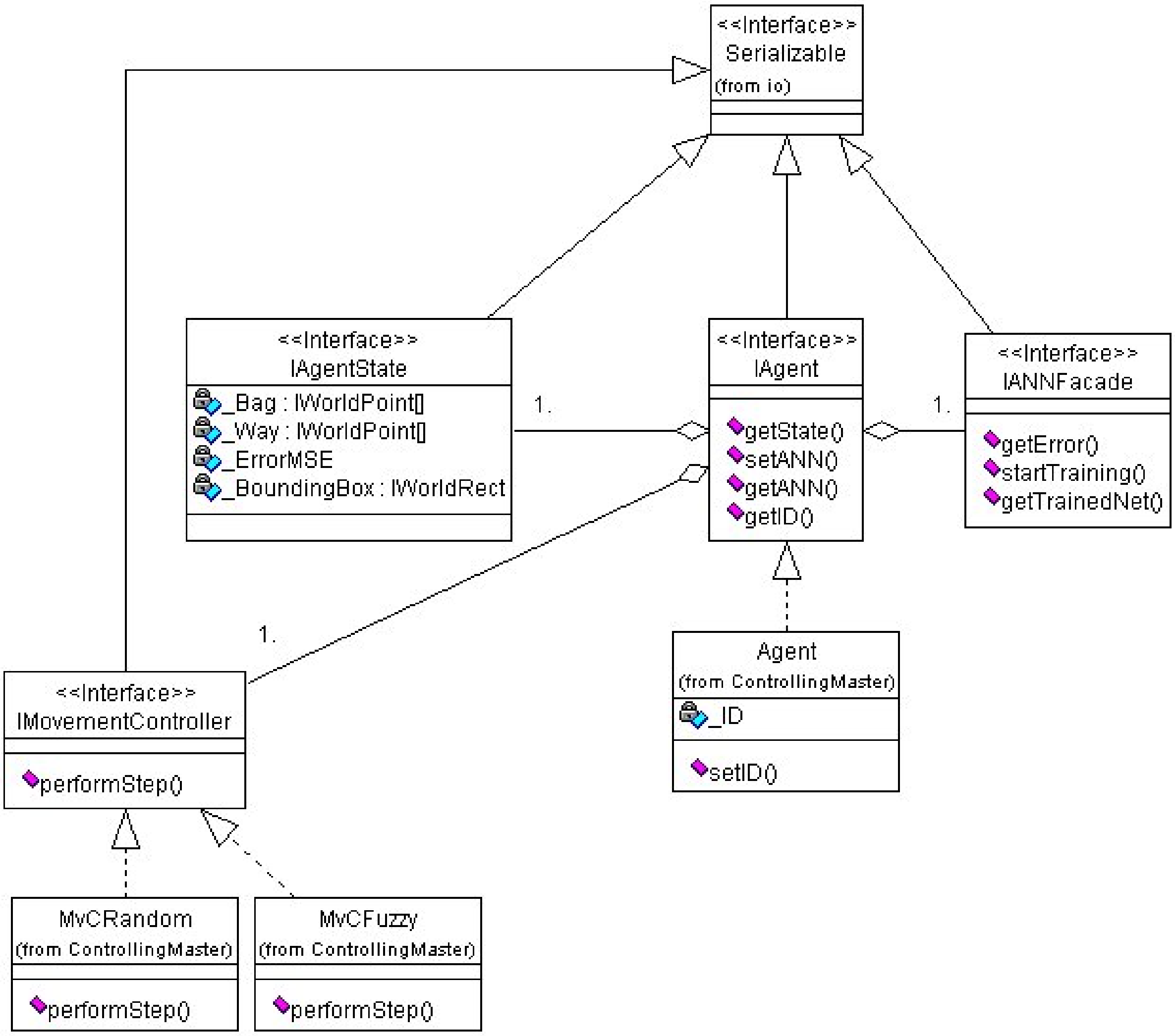

Figure 3.2: Class diagram: Agent Model

Special design decisions:

IMovementController This interface enables flexible replacement of movement behavior of an agent

IAgentState Agent state extracted from implementation to reduce network

3.3.2 Management Client

A smart and simple console interface is sufficient for this design stage (diagram 3.3).

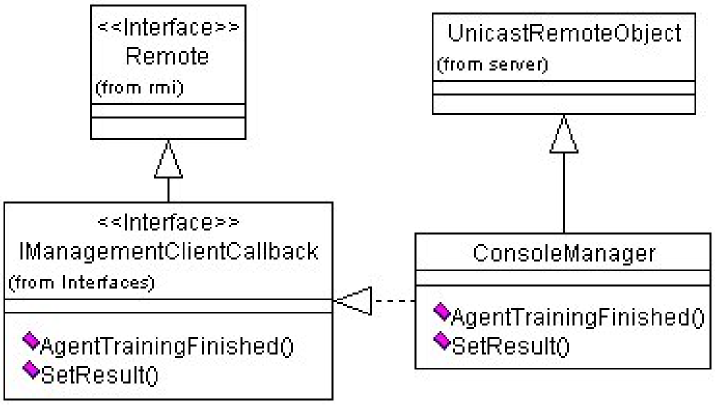

Figure 3.3: Class diagram: Manager Model

3.3.3 Controlling Master

Class modeling of the central intelligence component is in figure 3.4.

Special design decision:

IAgentFactory Separate modeling of an agent creation interface leads into a looser coupling between the agent controller and the working slave. Additionally variation of agent components is facilitated.

3.3.4 Working Slave

Figure 3.5 is a working slave's class diagram.

Special design decisions:

IANNFacade To minimize dependency on a distinct ANN-package, an abstraction level is introduced. Realized with the Facade design pattern from [15, pages 189-198].

JFroehlichANNFacade Due to lack of free available ANN-package Jörg Fröhlich wrote his own implementation, which fitted best for SatMAS at first glance.

3.3.5 World modeling

As discussed in section 2.2, best selection of a concrete world model needs to be researched. Therefore an abstract model for worlds and their data is needed, as seen in diagram 3.6.

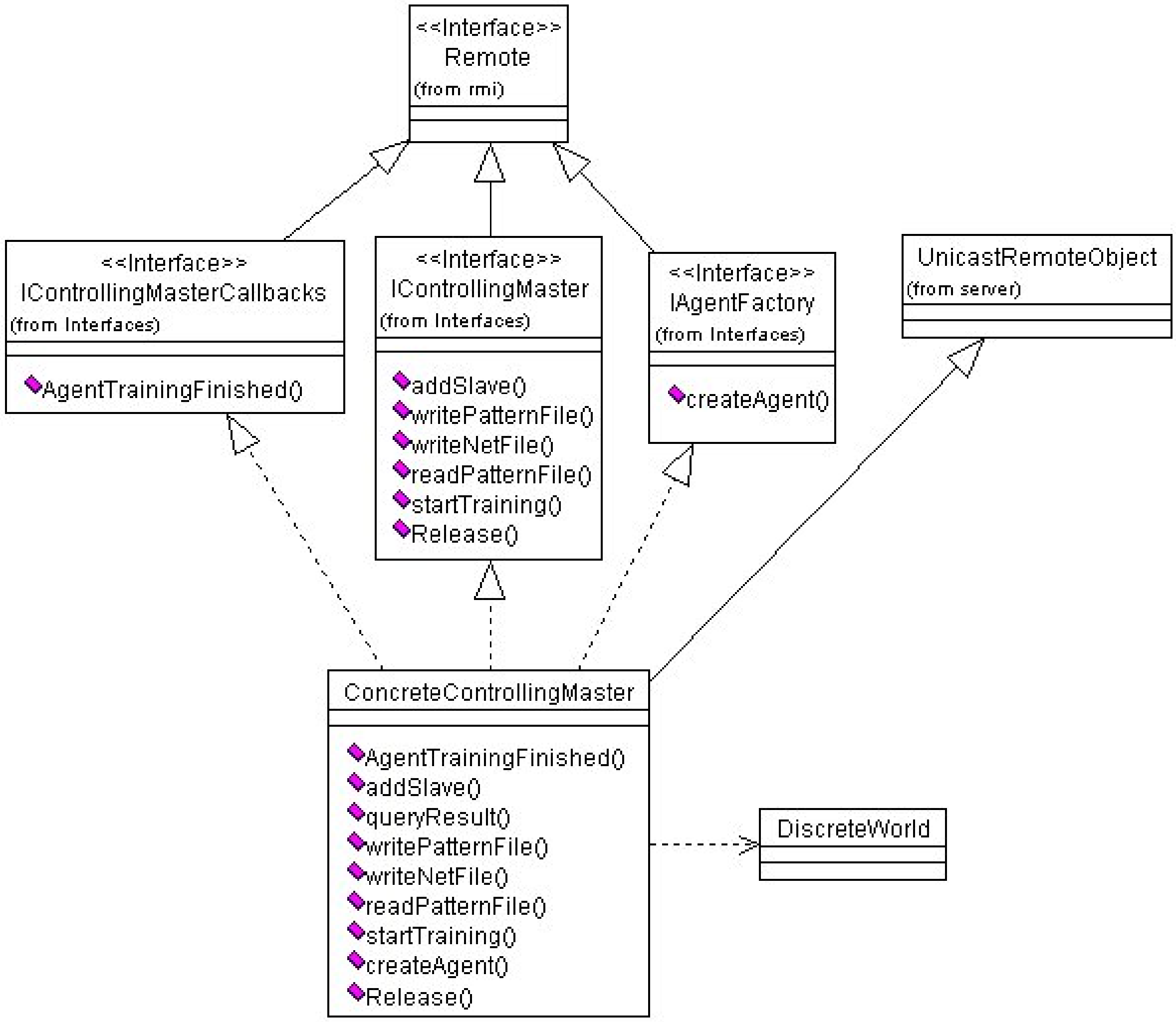

Figure 3.4: Class diagram: Master Model

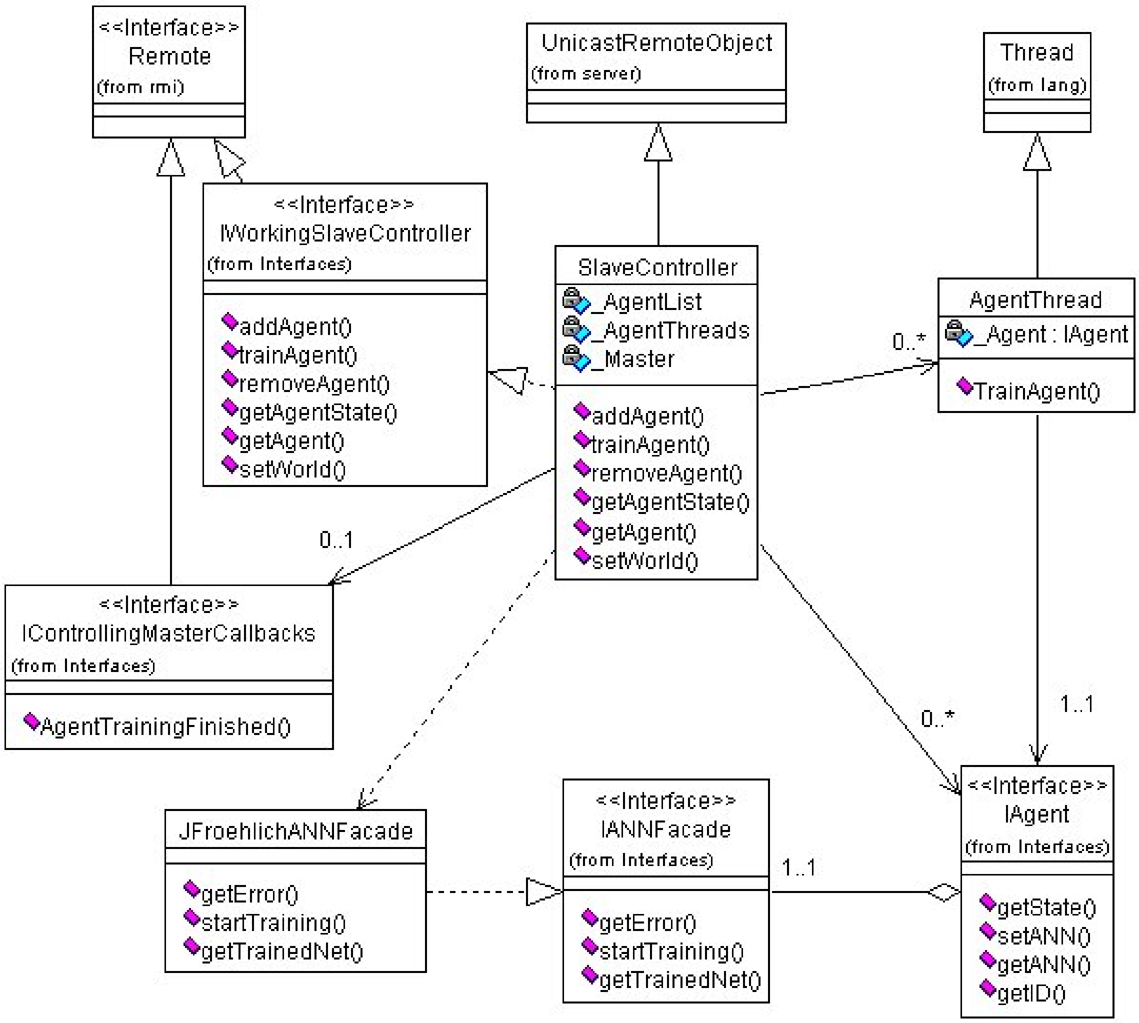

Figure 3.5: Class diagram: Slave Model

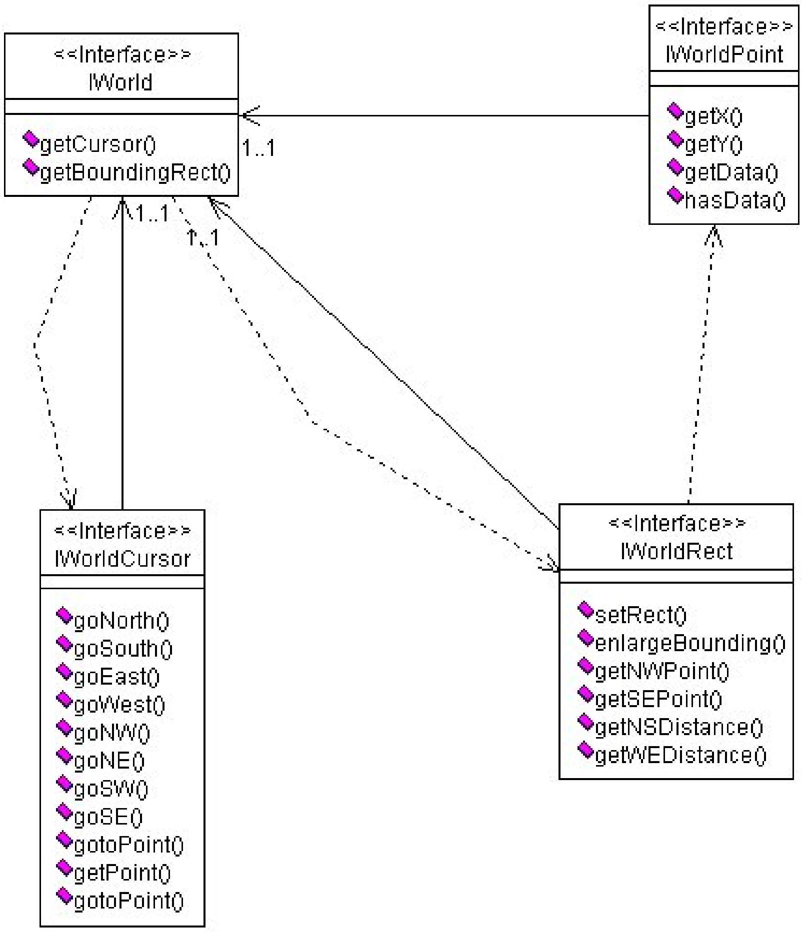

Figure 3.6: Class diagram: World Model

Special design decisions:

IWorld Abstract description of a real world model. After creation a world model is immutable.

IWorldPoint Represents a concrete position on a world model, including data access methods.

IWorldRect A rectangular region on a world model.

IWorldCursor Mobile position marker on a world model.

Concrete implementation of a world

This implementation example of a discrete world models a plane 2D area, on which each raster point is reachable.

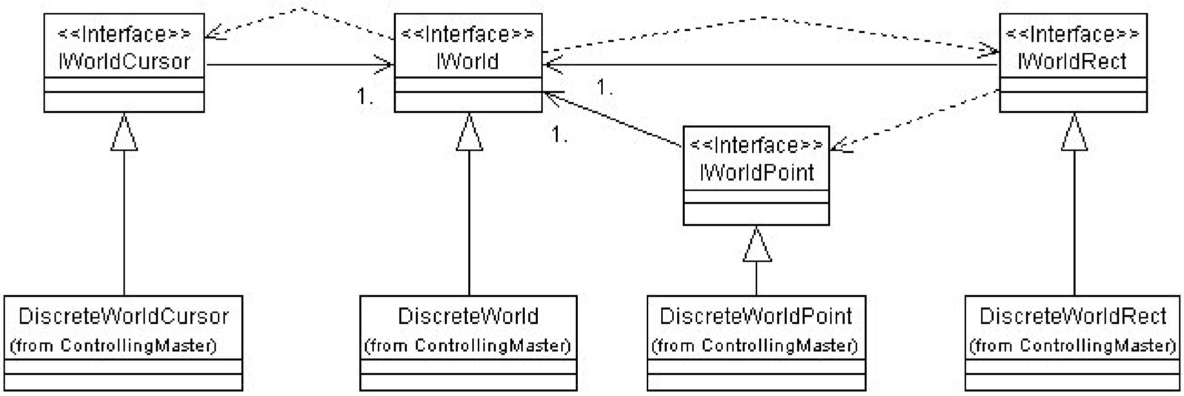

Figure 3.7: Class diagram: Concrete implementation of a world

3.4 Sequence diagrams

3.4.1 Agent training and result evaluation - overview

Diagram 3.8 gives a rough overview how agents are trained, and after finishing their learning phase the whole result is queried.

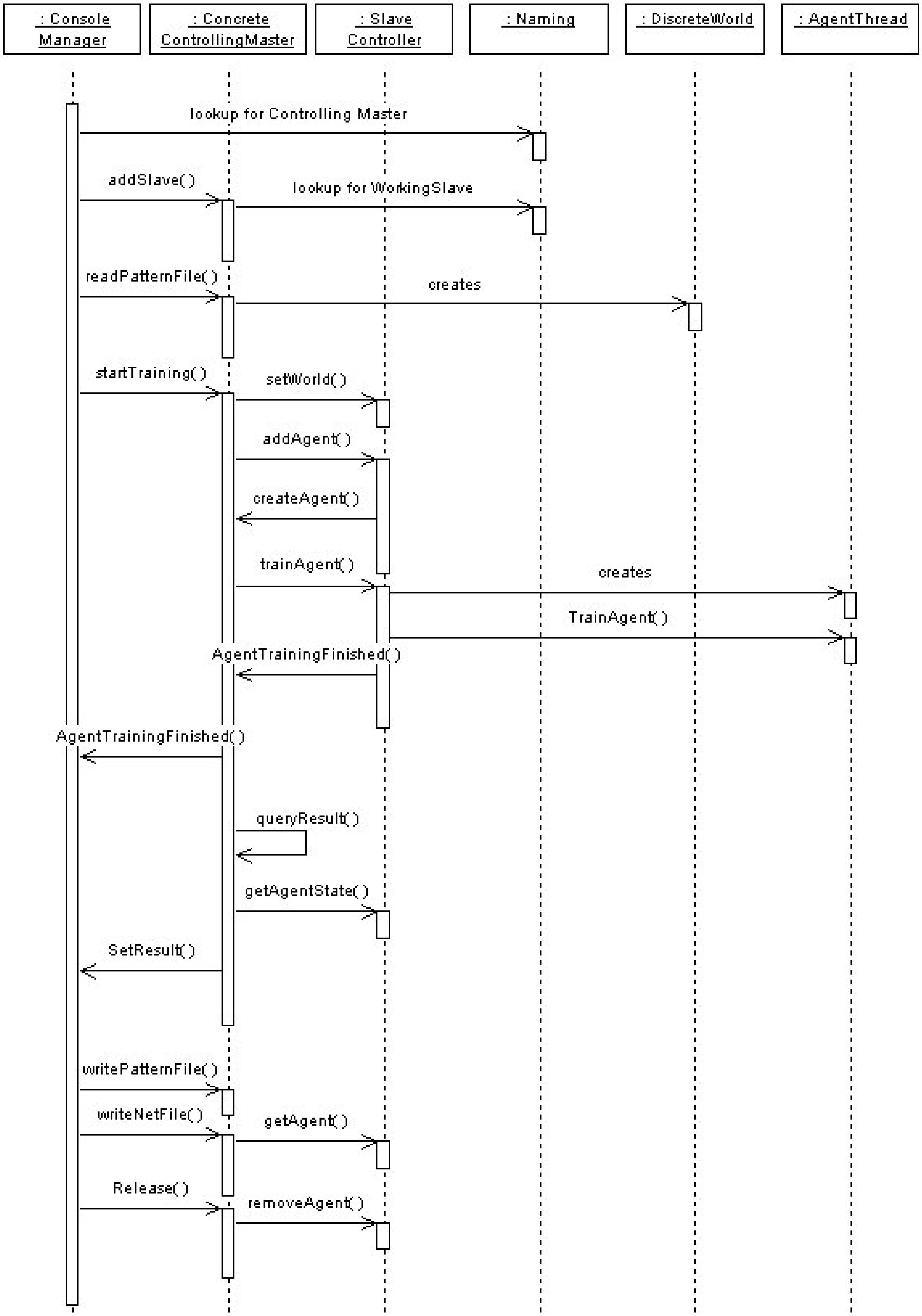

Figure 3.8: Sequence diagram: Agent training and result evaluation - overview

Chapter 4

Implementation and prototype testing

4.1 Implementation

Programming language: Java

JDK: only tested on 1.3

Neural Network package: jaNet

Pattern file library: NetJen

Unit test framework: JUnit

4.1.1 JESICA

A fancy name gives an excellent product the right touch. So we also had to find a agreeable-sounding name for this great java project. After weeks of hard, mind exhausting work, **JESICA** was born. JESICA is the short form of
J*ava* **E***xperimentation* **S***ytem for* **I***mage* **C***lassification with* **A***gents* .

4.2 First Run

Here comes the first pictures of JESICA's first working prototype.

Run parameter:

Number of Agents: 4

Input pattern size: 400 (20x20)

Input pattern type: XOR

Agent way length: 30

Agent movement step size: 2

Agent movement strategy: random

Agent bag size: 10

NN validation pattern set: Agent's way

NN minimum error: 0.001

NN training cycles: 1000

NN learning rate: 0.5

Run 1:

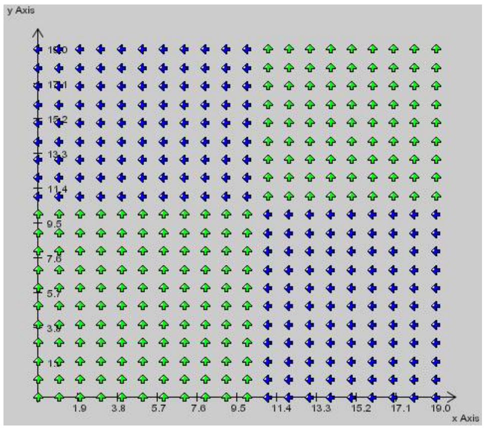

Figure 4.1: First Run: Input pattern

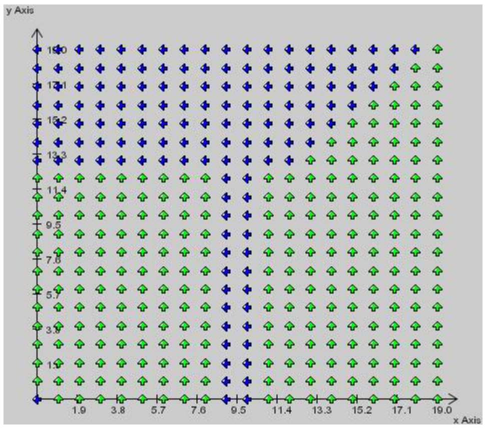

Figure 4.2: First Run: Result

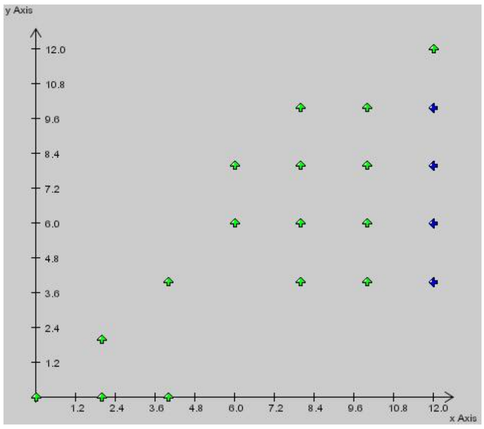

Figure 4.3: First Run: Way of first Agent

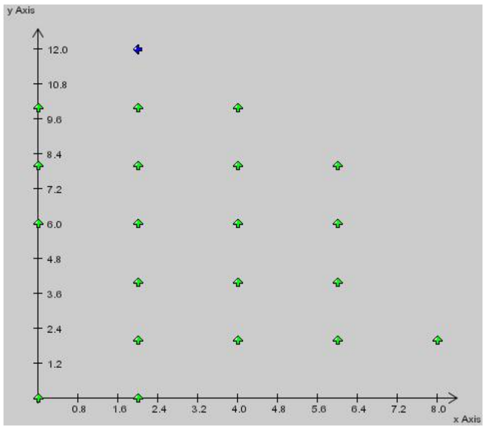

Figure 4.4: First Run: Way of second Agent

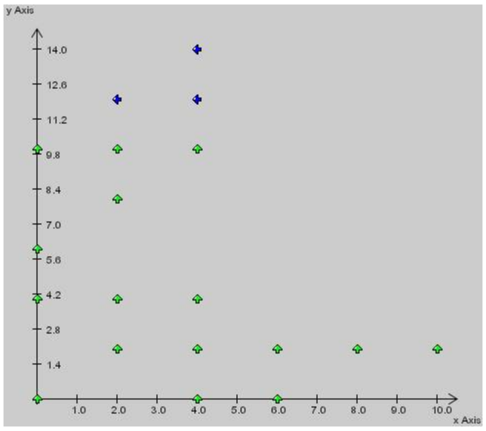

Figure 4.5: First Run: Way of third Agent

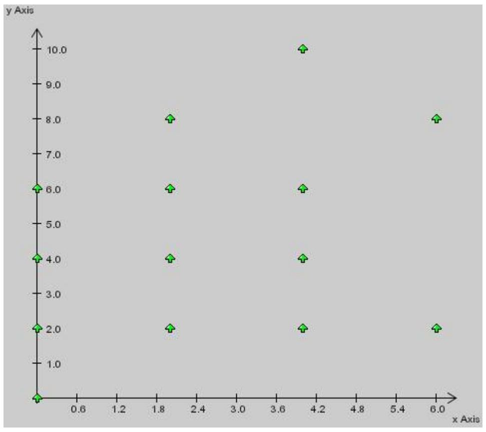

Figure 4.6: First Run: Way of fourth Agent

4.3 Further experiments with random agent movement

This agent's movement controller works on a simple, random based strategy. The individual movement steps will be decided by a randomizer, with the only constraint that no points are allowed that are already in the agent's bag.

Run parameter:

Number of Agents: 8

Input pattern size: 400 (20x20)

Input pattern type: XOR

Agent way length: 30

Agent movement step size: 2

Agent movement strategy: random, no points that are already in bag

Agent bag size: 10

NN validation pattern set: Agent's way

NN minimum error: 0.001

NN training cycles: 3000

NN learning rate: 0.1

Run 1:

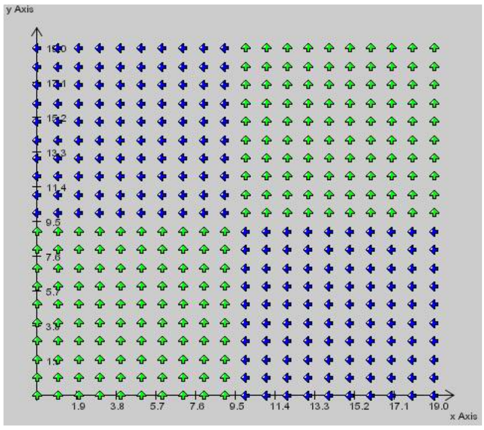

Figure 4.7: Random move: Input pattern

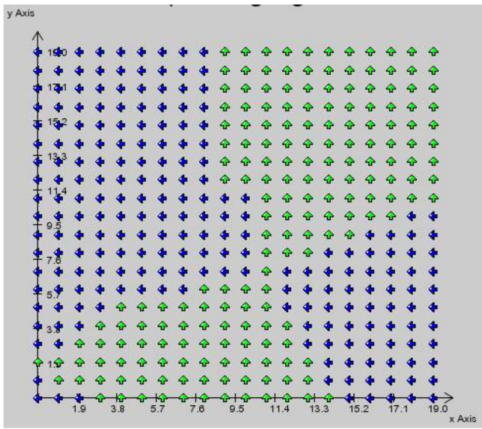

Figure 4.8: Random move, Run 1: Output pattern

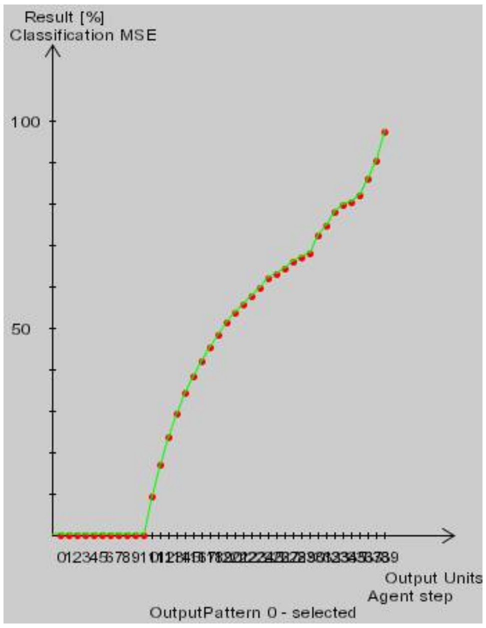

Figure 4.9: Random move, Run 1, Agent 1: Way MSE

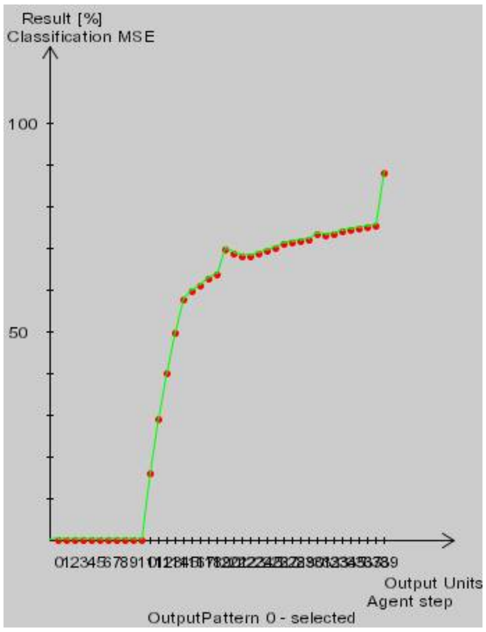

Figure 4.10: Random move, Run 1, Agent 2: Way MSE

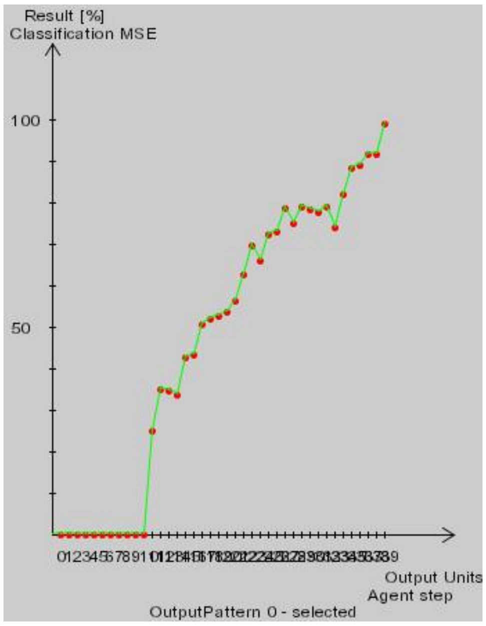

Figure 4.11: Random move, Run 1, Agent 3: Way MSE

4.3.1 Experiment: more Agents

Run parameter:

Number of Agents: 16

Input pattern size: 400 (20x20)

Input pattern type: XOR

Agent way length: 30

Agent movement step size: 2

Agent movement strategy: random, no points that are already in bag

Agent bag size: 10

NN validation pattern set: Agent's way

NN minimum error: 0.001

NN training cycles: 3000

NN learning rate: 0.1

Run 2:

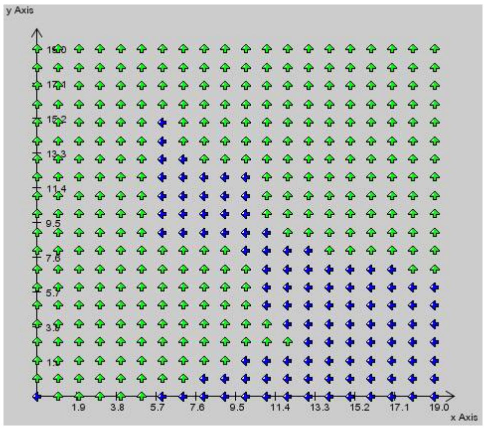

Figure 4.12: Random move, Run 2: Output pattern

4.3.2 Experiment: smaller step size

Run parameter:

Number of Agents: 16

Input pattern size: 400 (20x20)

Input pattern type: XOR

Agent way length: 30

Agent movement step size: 1

Agent movement strategy: random, no points that are already in bag

Agent bag size: 10

NN validation pattern set: Agent's way

NN minimum error: 0.001

NN training cycles: 3000

NN learning rate: 0.1

Run 3:

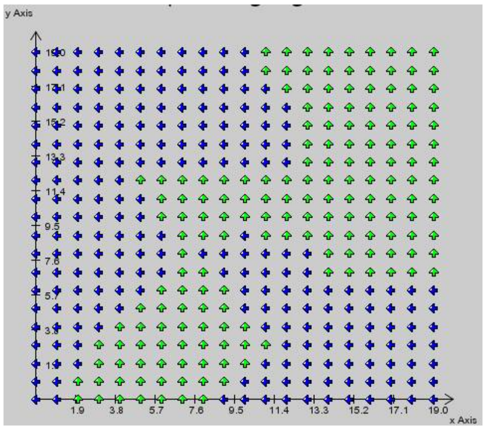

Figure 4.13: Random move, Run 3: Output pattern

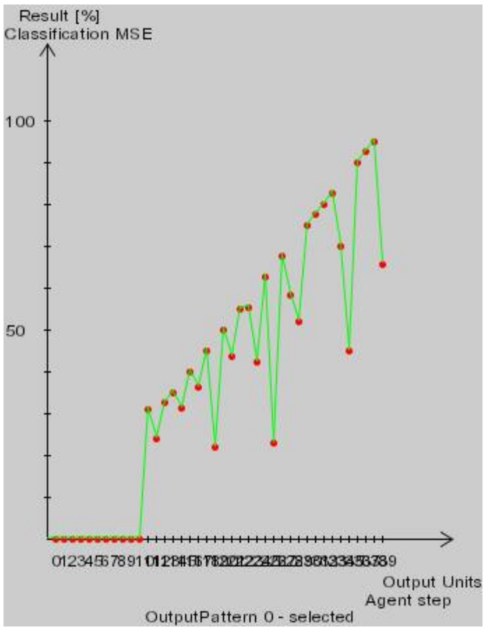

Figure 4.14: Random move, Run 3, Agent 1: Way MSE

4.4 Experiments with centroid agent movement

This agent implementation uses two different strategies for movement:

- If the gradient (calculated over a specified period) of the way MSE is less or equal null, random movement is used.
- If the gradient is positive, a centroid movement strategy will be used. Here the vectors of all point in the agent's bag will be calculated, and the resulting direction

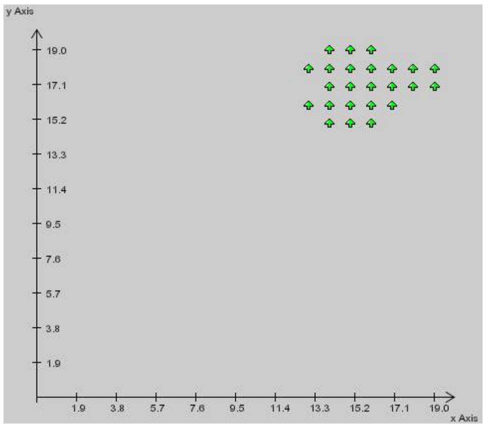

Figure 4.15: Random move, Run 3, Agent 1: Way

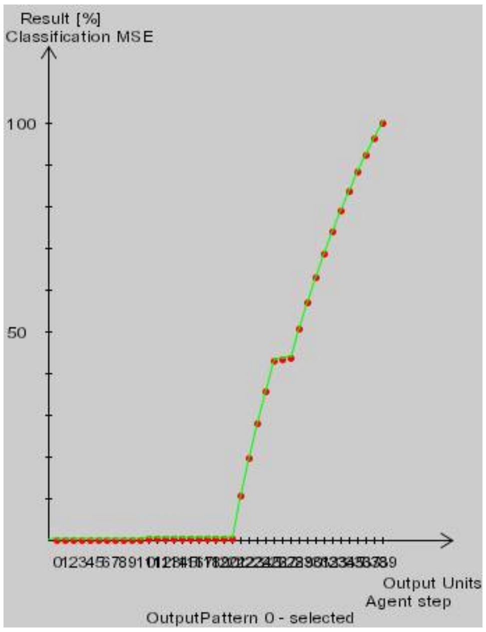

Figure 4.16: Random move, Run 3, Agent 2: Way MSE

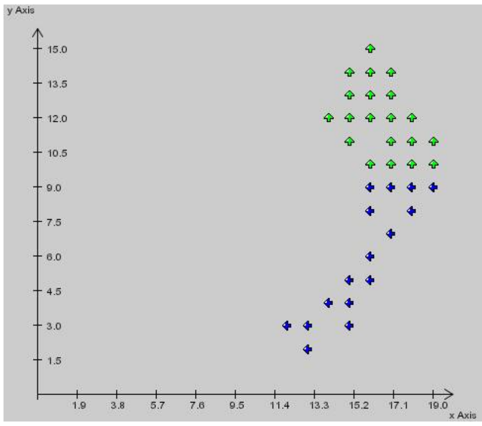

Figure 4.17: Random move, Run 3, Agent 2: Way

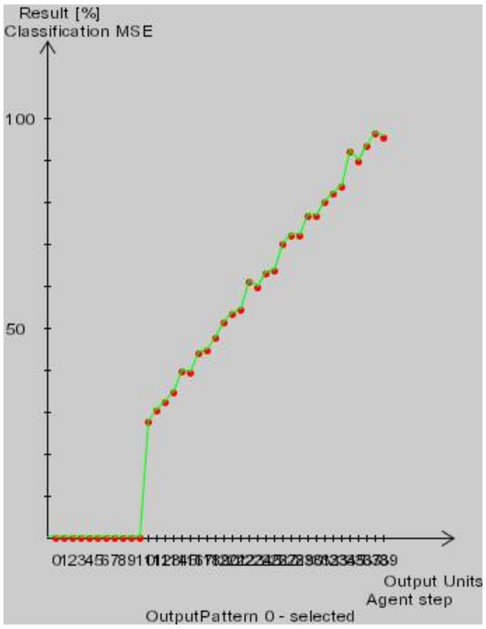

Figure 4.18: Random move, Run 3, Agent 3: Way MSE

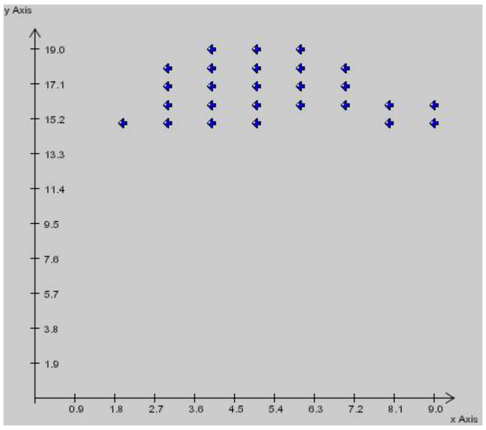

Figure 4.19: Random move, Run 3, Agent 3: Way

will be used as parameter for a "roulette-wheel-random-number generator". This generator is realized by using a modified Gaussian probability spreading.

Run parameter:

Number of Agents: 8

Input pattern size: 400 (20x20)

Input pattern type: XOR

Agent way length: 30

Agent movement step size: 2

Agent movement strategy: random, centroid based error gradient positive

Gradient calculation threshold: 5

Agent bag size: 10

NN validation pattern set: Agent's way

NN minimum error: 0.001

NN training cycles: 3000

NN learning rate: 0.1

Run 1

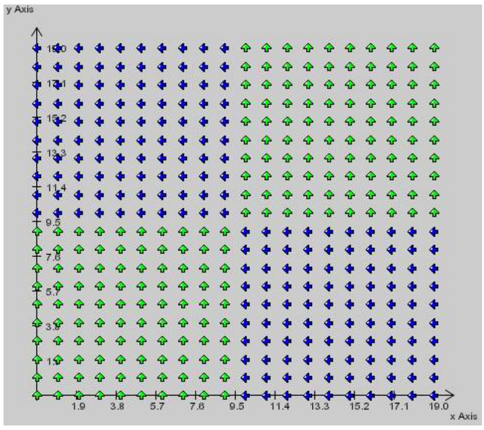

Figure 4.20: Centroid move: Input pattern

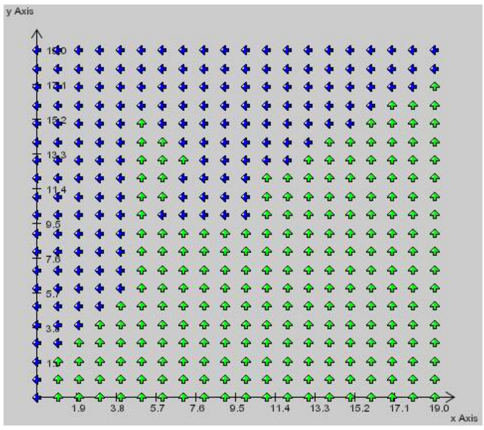

Figure 4.21: Centroid move, Run 1: Output pattern

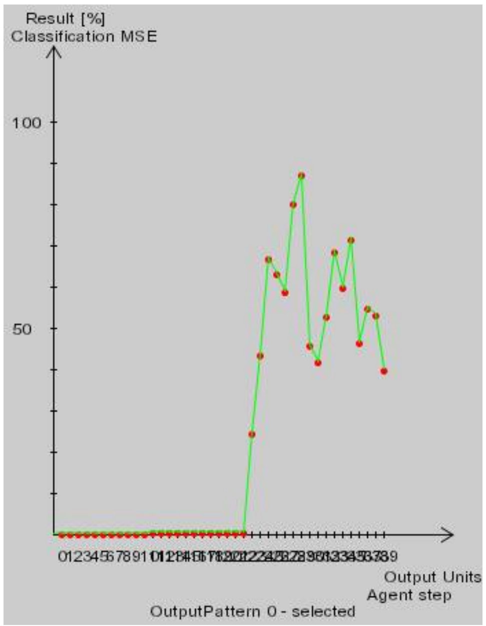

Figure 4.22: Centroid move, Run 1, Agent 1: Way MSE

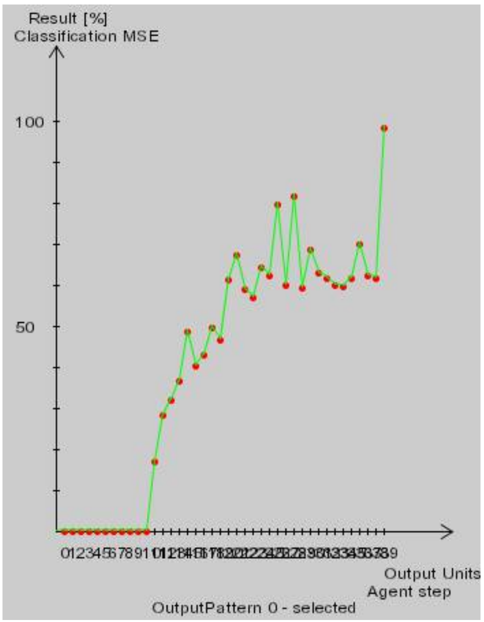

Figure 4.23: Centroid move, Run 1, Agent 2: Way MSE

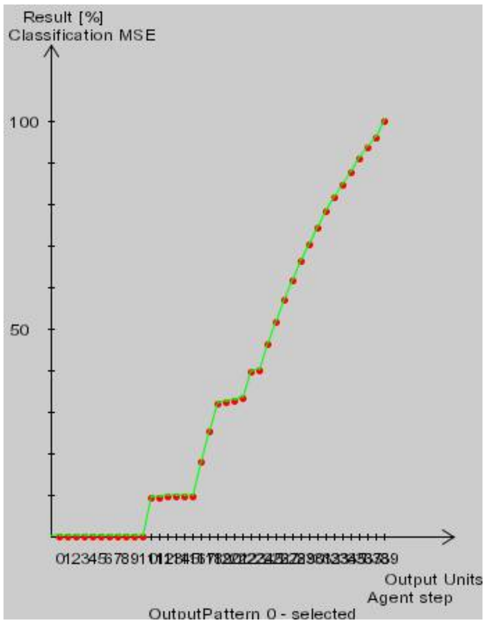

Figure 4.24: Centroid move, Run 1, Agent 3: Way MSE

Run 2

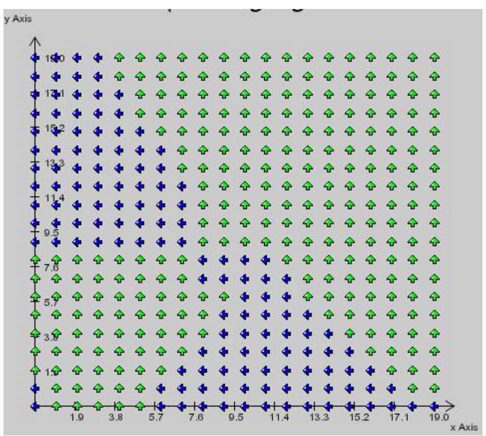

Figure 4.25: Centroid move, Run 2: Output pattern

Run 3

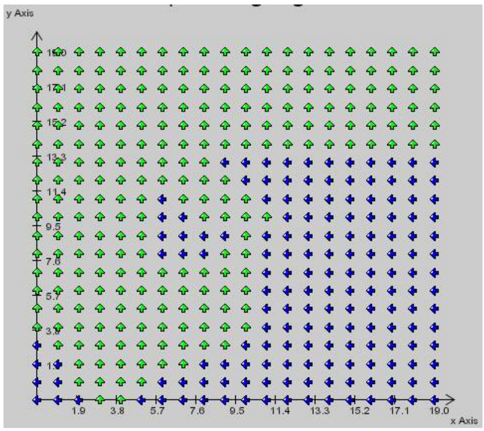

Figure 4.26: Centroid move, Run 3: Output pattern

4.4.1 Experiment: more Agents, smaller step size

Run parameter:

Number of Agents: 16

Input pattern size: 400 (20x20)

Input pattern type: XOR

Agent way length: 30

Agent movement step size: 1

Agent movement strategy: random, centroid based error gradient positive

Gradient calculation threshold: 5

Agent bag size: 10

NN validation pattern set: Agent's way

NN minimum error: 0.001

NN training cycles: 3000

NN learning rate: 0.1

Run 4

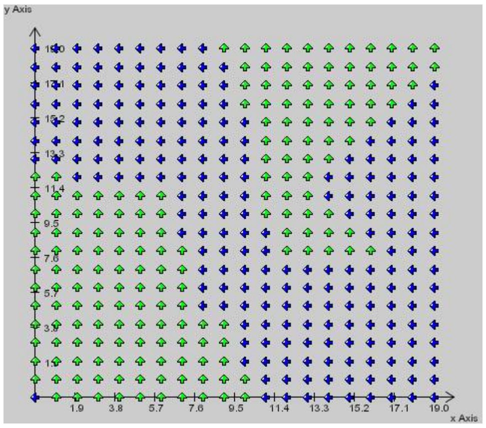

Figure 4.27: Centroid move, Run 4: Output pattern

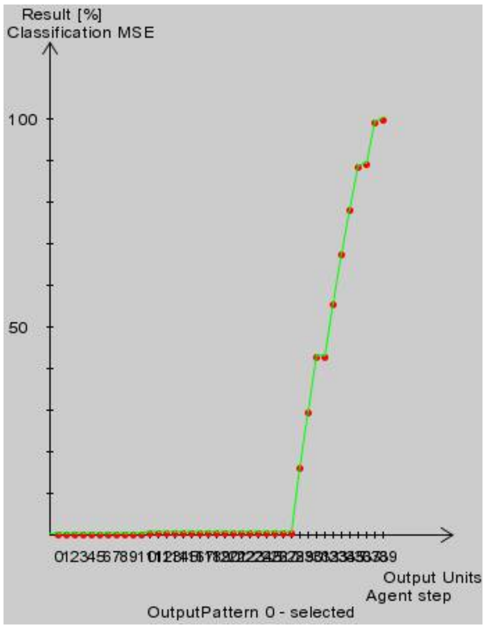

Figure 4.28: Centroid move, Run 4, Agent 1: Way MSE

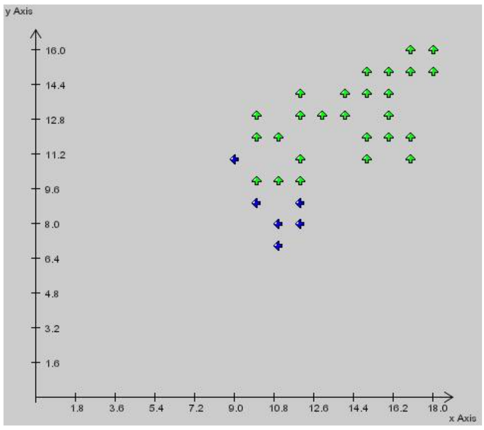

Figure 4.29: Centroid move, Run 4, Agent 1: Way

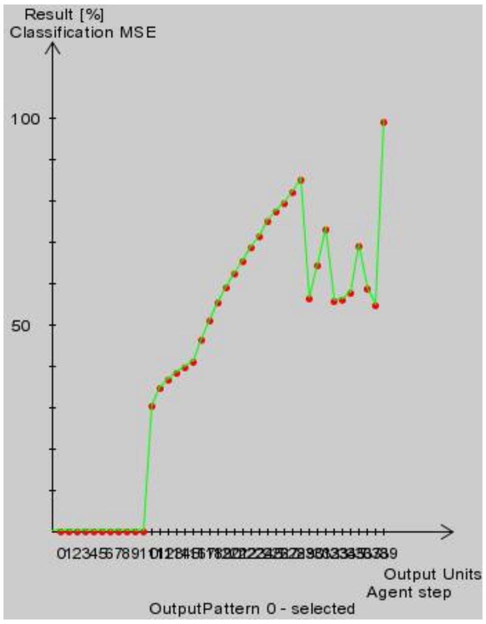

Figure 4.30: Centroid move, Run 4, Agent 2: Way MSE

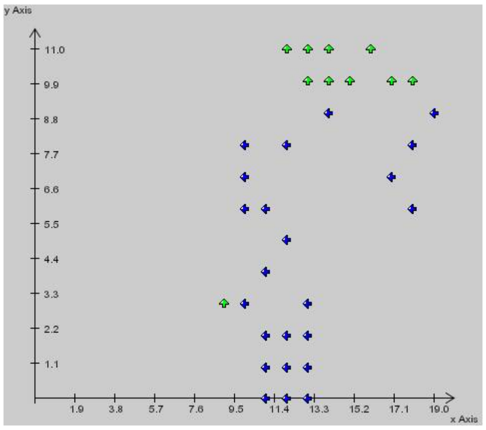

Figure 4.31: Centroid move, Run 4, Agent 2: Way

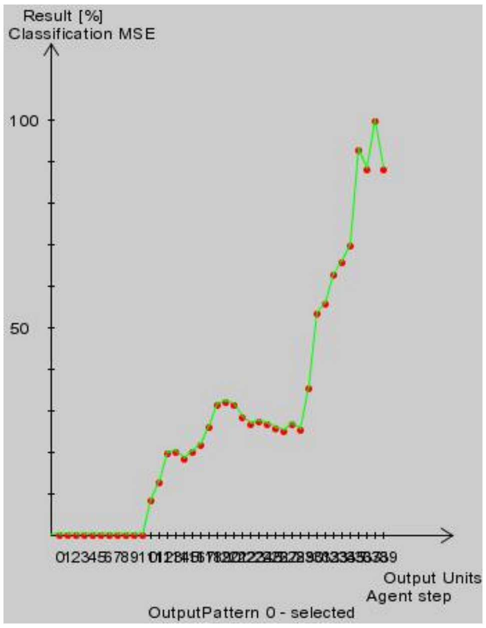

Figure 4.32: Centroid move, Run 4, Agent 3: Way MSE

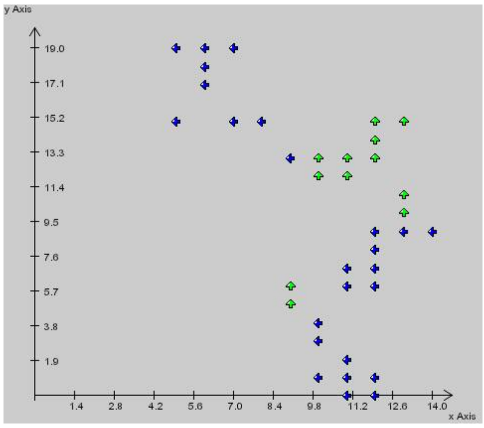

Figure 4.33: Centroid move, Run 4, Agent 3: Way

Run 5

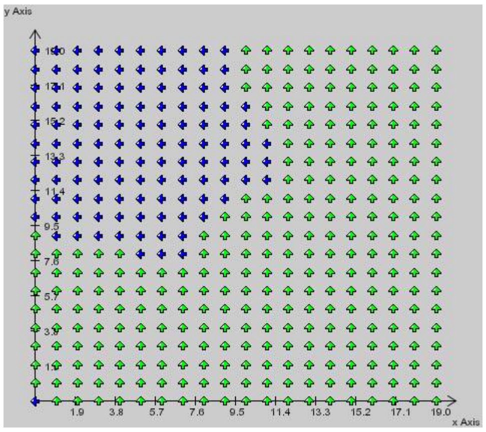

Figure 4.34: Centroid move, Run 5: Output pattern

4.4.2 Experiment: smaller gradient threshold

Run parameter:

Number of Agents: 16

Input pattern size: 400 (20x20)

Input pattern type: XOR

Agent way length: 30

Agent movement step size: 1

Agent movement strategy: random, centroid based error gradient positive

Gradient calculation threshold: 3

Agent bag size: 10

NN validation pattern set: Agent's way

NN minimum error: 0.001

NN training cycles: 3000

NN learning rate: 0.1

Run 6

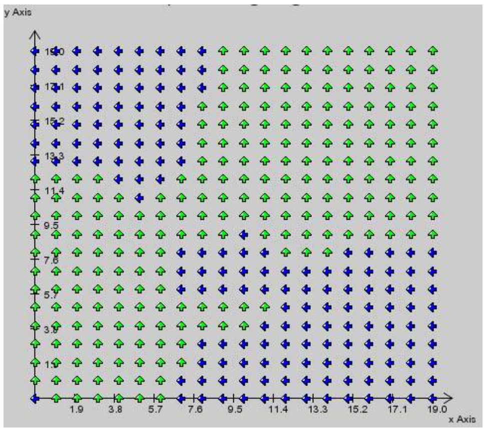

Figure 4.35: Centroid move, Run 6: Output pattern

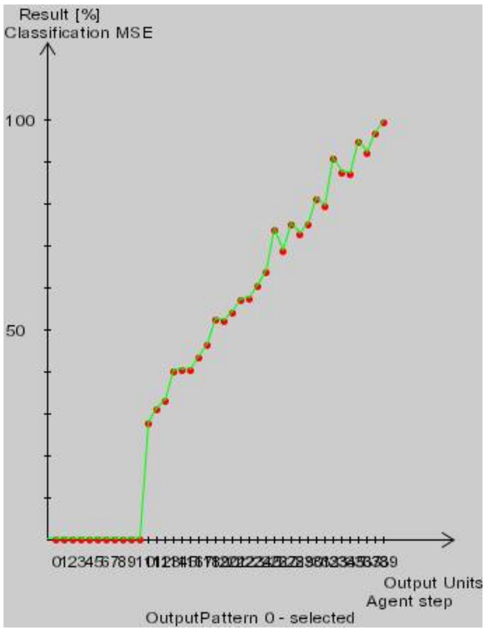

Figure 4.36: Centroid move, Run 6, Agent 1: Way MSE

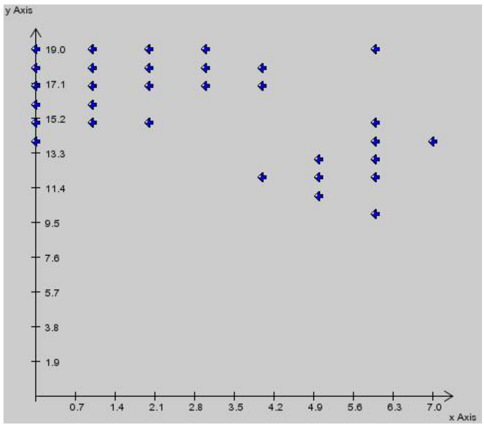

Figure 4.37: Centroid move, Run 6, Agent 1: Way

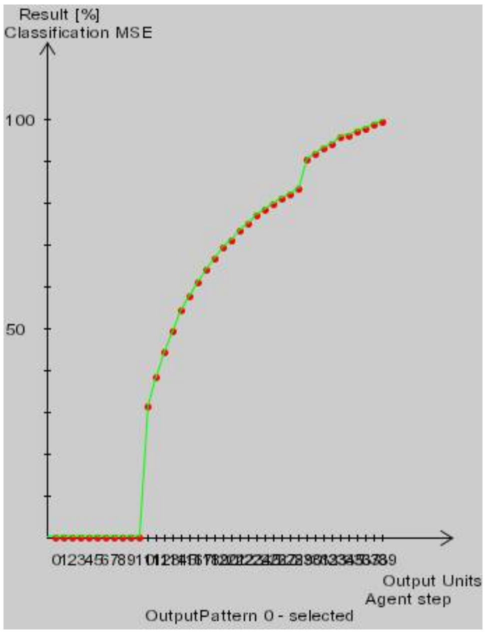

Figure 4.38: Centroid move, Run 6, Agent 2: Way MSE

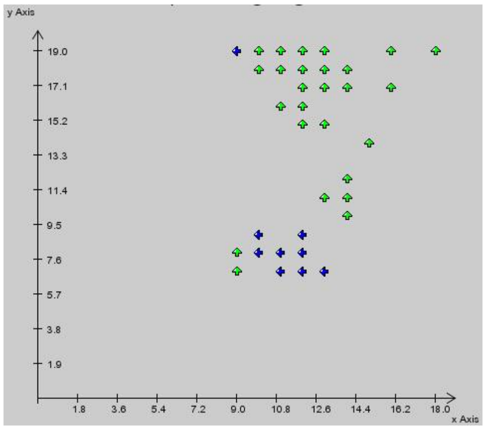

Figure 4.39: Centroid move, Run 6, Agent 2: Way

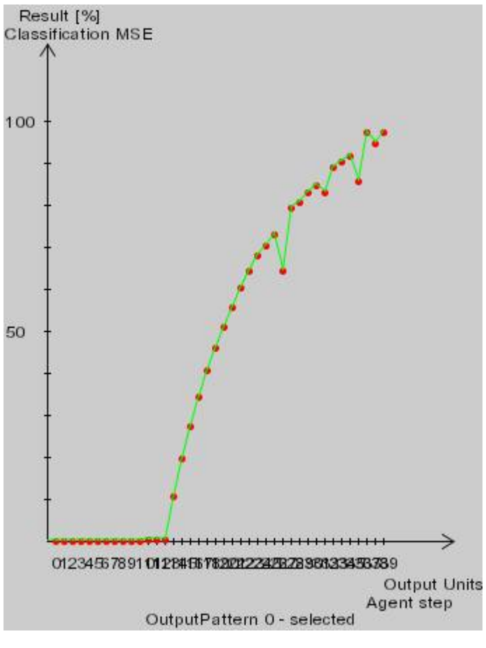

Figure 4.40: Centroid move, Run 6, Agent 3: Way MSE

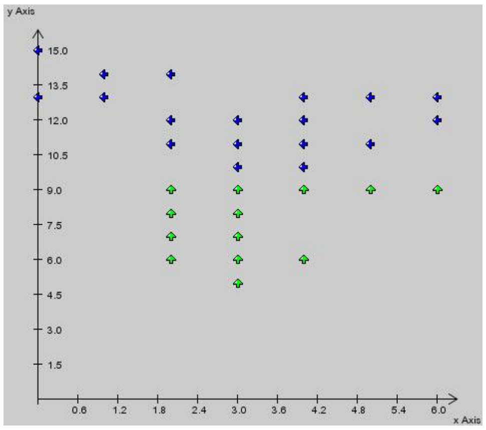

Figure 4.41: Centroid move, Run 6, Agent 3: Way

Run 7

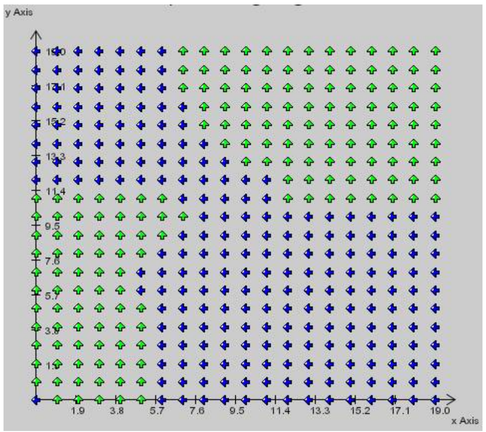

Figure 4.42: Centroid move, Run 7: Output pattern

Bibliography

[1] Doron Shalvi. An introduction to artificial neural networks, 1997.

[2] Perry Moerland. Classification using localized mixtures of experts. In *Proceedings of the International Conference on Artificial Neural Networks (ICANN'99)*, volume 2, pages 838–843. London: IEE, 1999. (IDIAP-RR 98-14).

[3] Victor R. Lesser. Cooperative multiagent systems: A personal view of the state of the art. In *IEEE Transaction on Knowledge and Data Engineering*, volume 11, No. 1 Jan/Feb 1999, pages 133–142, 1999.

[4] Frank Buschmann; Regine Meunier; Hans Rohnert; Peter Sommerlad; Michael Stal. *Pattern-orientierte Software-Architektur. Ein Pattern-System.* Addison-Wesley, 1st edition, 1998. ISBN 3-8273-1282-5.

[5] H. Ritter M. Hasenjäger. Active learning in neural networks. Technical report, Technische Fakultät, Universität Bielefeld, Germany, 1998.

[6] Leonard G.C. Hamey Tirthankar RayChaudhuri. Accurate modelling with minimised data collection – an active learning algorithm. Technical report, School of MPCE, Macquarie University, New South Wales 2109, Australia, 1995.

[7] Axel Röbel. The dynamic pattern selection algorithm: Effective training and controlled generalization of backpropagation neural networks. Technical report, TU Berlin, Germany, 1998.

[8] Helmut Mayer Roland Schwaiger. Is evolution a good teacher? genetic algorithms create training data sets for artificial neural networks. Technical report, Department of Computer Science, University of Salzburg, A-5020 Salzburg, Austria, 1998.

[9] Kenji Nakayama Kazuyuki Hara. Training data selection method for generalization by multilayer neural networks. Technical report, The Institute of Electronics, Information and Communication Engineers, Tokyo 105, Japan, 1998.

[10] Third untited nations conference on the exploration and peaceful use of outer space, 1998. A/CONF.184/BP/3.

[11] Stephen Quirolgico; Kip Canfield; Timothy Finnin; James A. Smith. Communicating neural network knowledge between agents in a simulated aerial reconnaissance system. Technical report, Department of Information Systems, University of Maryland. Baltimore County, 2000.

[12] Bjarne Stroustrup. *Die C++ Programmiersprache.* Addison-Wesley, 3rd edition, 1998. ISBN 3-8273-1296-5.

[13] Rainer Burkhardt. *UML - Unified Modeling Language. Objektorienntierte Modellierung für die Praxis.* Addison-Wesley, 2nd edition, 1999. ISBN 3-8273-1407-0.

[14] Erich Gamma; Richard Helm; Ralph Johnson; John Vlissides. *Design Patterns - Elements of reusable object oriented software.* Addison-Wesley, 1st edition, 1995. ISBN 0-201-63361-2.

[15] Erich Gamma; Richard Helm; Ralph Johnson; John Vlissides. *Entwurfsmuster - Elemente wiederverwendbarer objektorientierter Software.* Addison-Wesley, 1st edition, 1996. ISBN 3-89319-950-0.