INTEGRATING MOBILE AGENTS AND NEURAL NETWORKS FOR PROACTIVE MANAGEMENT

Klaus Herrmann, Kurt Geihs
Johann Wolfgang Goethe-University
Department of Computer Science
Robert-Mayer-Str. 1
60325 Frankfurt/Main, Germany
{klaus,geihs}@vsb.cs.uni-frankfurt.de

Abstract The management of modern computer networks and distributed systems comprises big challenges due to the complexity of nowadays network environments. It requires a decentralized approach in order to be efficient, effective, scalable, and flexible. Moreover, it has to be proactive since identifying and preventing problems *before* they affect users becomes increasingly important. Several research projects have identified mobile agents as a possible solution for decentralized management, while others have used neural networks to make predictions and achieve proactiveness. We propose a management system which integrates both technologies to achieve proactiveness in a decentralized fashion. The result is a distributed management system that employs intelligent mobile components. This paper discusses our approach and presents the design and implementation of a prototype application that puts the proposed ideas to practice.

Keywords: Mobile Agents, Network Management, Proactive Management, Neural Networks, Prediction

1. INTRODUCTION

With the rapid growth of modern computer networks the management of these networks and their distributed applications has gained strategic importance. Moreover, new trends like the integration of fixed corporate networks and wireless terminal devices like laptops and palmtops introduce new levels of complexity for network management. These tendencies towards larger networks and more complexity and heterogeneity have triggered two development strands in distributed system management which are still only partially connected:

- **Decentralization:** By distributing the management intelligence within the managed network the system gains flexibility, scalability, and robustness.
- **Intelligent data analysis tools:** Manual analysis of management data from large, heterogeneous networks has become impractical or even impossible. By employing intelligent techniques, relevant information can be extracted from large sets of seemingly unstructured data.

While there are many research projects that either concentrate on decentralizing network management or on using intelligent analysis methods, the integration of both aspects has been rarely studied yet. This is mainly due to several conflicting properties of large scale intelligent systems and flexible decentralization mechanisms.

In this paper we will discuss how *mobile agents* (MAs) can employ *artificial neural networks* (NNs) to achieve intelligent data analysis in a decentralized fashion. In section 2 we describe the requirements and examine MAs and NNs with respect to network management. Section 3 will motivate the integration of MAs and NNs and discuss general technical challenges of this approach. We give a concrete design and analyze its properties. Section 4 introduces the IntraManager system which is an implementation of the proposed design for the management of IP networks. Finally, sections 5 and 6 will present related work and conclusions.

2. REQUIREMENTS AND TECHNOLOGIES

2.1. Decentralized and Proactive Management

When IAB and ISO developed their original network management (NM) standards more than a decade ago they assumed that a centralized approach would be appropriate to solve most NM problems. In these frameworks one or more central *manager* applications communicate with a set of *NM agents* that are installed on the managed nodes and serve as a data base for NM information. Putting intelligence on the managed nodes was not an option at that time because a small footprint approach was needed. Thus, these management frameworks did not include facilities to dynamically decentralize NM intelligence within the managed network.

The NM community quickly realized that NM had to be decentralized to handle the growing demands. The basic idea behind this decentralization is to express a NM task as a piece of *mobile code* [Baldi et al., 1997] which is sent to a managed device to monitor and control it locally. Decentralization can lead to some considerable improvements [Goldszmidt, 1993]. *Micro management* can be avoided, i.e. sequentially issued management commands can be encapsulated into one mobile code entity and need not be transmitted separately. Instead of collecting raw NM data to analyze it at the central location, mobile NM code can be sent to the devices to process the data where it originates. This leads to much more scalable applications because the processing load and network load are distributed over the whole network. A notable increase in robustness is achieved by avoiding *central points of failure* and *bottlenecks*. Moreover, an NM application based on mobile code is adaptable to dynamic changes within the managed network by adding or removing NM functions at managed nodes on the fly during runtime.

Today, the necessity for decentralization in NM standards seems to be generally agreed upon. Projects like the RMON MIB [Waldbusser, 2000] and DISMAN [Levi and Schoenwaelder, 1999] within the IETF (*Internet Engineering Task Force*), and the research initiated by Goldszmidt and Yemini with their *Management by Delegation* approach (MbD) [Goldszmidt, 1993] prove this fact. MbD was the first project that made use of *mobile code* to delegate NM tasks to managed nodes. Other researchers (e.g. [Kooijman, 1995], [Zapf et al., 1999], [El-Darieby and Bieszczad, 1999]) followed the same principle using different technologies.

Besides decentralization, another NM topic has gained considerable attention as managed networks continue to grow in size, complexity and importance. Such networks produce vast amounts of NM data which makes intelligent data analysis technologies a necessity. Furthermore, administrators seek for *proactive* NM tools that help to detect and prevent problems beforehand in order to avoid down times and financial losses.

A logical consequence of these observations is the integration of both decentralization and proactiveness into one framework. Such a framework should allow proactive, mobile NM tasks to manage the nodes of a network autonomously, i.e. without constant interventions from a central manager. To achieve the decentralization we chose mobile agents (MAs), while the aspect of proactiveness can only be realized by applying some intelligent prediction method that enables a NM tool to plan *one step ahead*. In our case we needed a very flexible mechanism in order to avoid frequent manual calibration which would interfere with the idea of autonomous decentralized NM tasks. Automatic adaptation to new and changing patterns in the measured data was important. Therefore, we chose artificial neural networks (NNs) as a prediction mechanism.

The next two sections will discuss MAs and NNs in more detail and motivate our decision for these technologies.

2.2. Intelligent Mobile Agents

Software agents are *self-contained*, *autonomous* software entities that can *perceive their environment*, *react to it in a timely fashion* and possibly change it according to their *goals*. They exhibit *goal-directed behavior* and thus are *proactive.* Agents can *interact with each other* through some kind of common *agent communication language.* They *act on behalf* of some human being or another software entity. In addition agents are called *mobile* if they are able to *migrate* to other hosts. Depending on the mechanisms that determine their bevahior, they may also be called *intelligent* (according to [Wooldridge and Jennings, 1995]).

Mobile intelligent agents appear to be a very attractive paradigm for complex, dynamic and distributed software. Agents are self-contained and generally use a message-based communication scheme. This leads to a highly flexible and modular approach. In addition, the concept of autonomy facilitates modular software systems whose modules (agents) are mobile and only loosely coupled. Therefore, a high degree of robustness can be introduced when designing agent-based software. Autonomy is mainly achieved through giving agents their own flow of control which enables them to actively take decisions.

All of the described abilities make mobile intelligent agents a good choice for decentralized NM (compare section 2.1). NM tasks can be represented as agents. When such an agent is mobile it becomes a *mobile manager* acting on behalf of an entity situated on a higher hierarchical level.

In the design of our own agent-based NM framework [Zapf et al., 1999] we use the notion of *health agents.* Such an agent monitors and controls a certain aspect of the *health* [Goldszmidt, 1993] of one or more managed nodes. Health agents are mobile but possess only a limited mobility scope. They can be viewed as intelligent agents that exploit their mobility for initial distribution and dynamically changing the degree of NM decentralization. After their migration to a primary target host they may autonomously react to changes in the network (or the NM policy) by moving away from or closer to their target host(s). As a reaction to the observed state they can spawn other MAs which may use their mobility to fix problems.

As we show in [Zapf et al., 1999], the use of well-known management protocols, in particular SNMP, allows for a very flexible distribution of health agents within a given network of nodes. These protocols are used by health agents to collect data. An agent may be put directly on a managed node and use SNMP locally or, if the node does not allow the execution of agent software, it may manage the device from a nearby node by using SNMP remotely over *short distances.* This eliminates the need for an agent execution environment on every managed node and helps reduce the impact of management.

2.3. Artificial Neural Networks

An artificial neural network (NN) presents a simple computational model of a biological neural system. In a NN, a number of primitive computing units (the neurons) are arranged in interconnected layers. In a *feed-forward* NN each neuron of one layer is connected to every neuron of the next layer. Each connection has a weight attached to it, that is adjusted during a training process. Every neuron calculates the weighted sum over its inputs and filters this sum through a non-linear *activation function*[1]. The result is propagated through its output connections and may serve as input for other neurons. After training such a NN with examples from a data set, the *general structure* of the data manifests itself in the network's weights. When deployed to *new* data from the same source it can still isolate this structure, even in the presence of noise, i.e. the NN can *generalize.* It reproduces the mapping learned during the training phase. Their ability to learn many different concepts with just one computational model, their flexibility to adapt to changes in the input data, and the efficiency of trained NNs make them so attractive. NNs are often applied to data analysis problems of which we only have a somewhat blurred understanding and which are therefore hard to solve with *classical* mathematical models.

Gardner and Harle show in [Gardner and Harle, 1997] how *Kohonen self-organizing maps* – a special kind of NN – can be used to map sets of generated NM events to their root causes (*event correlation*). In [Jobmann et al., 1997] the authors make use of a simple 2-layer feed-forward NN to correlate alarms in cellular phone networks. A second very appealing application, which we will focus on in this paper, is *time series prediction.* As was shown in [Edwards et al., 1997] a 2-layer feed-forward NN can be successfully applied for predicting the traffic in a network. In [Biesterfeld et al., 1997] the authors use the same technique to predict the future location of a mobile device within a cellular network. The setup used in these cases is also adopted for our work: The NM parameter which is to be predicted is periodically measured at discrete time intervals and thus can be expressed as a time series $\{x_0, x_1, \ldots, x_k\}$ where x_i is the parameter's value at time index i and k is the time index of the most recent measurement. The goal is to predict x_{k+1} given the window of values $\{x_{k-n-1}, \ldots, x_k\}$. For example, a NN could take the last 12 hourly system load values $(n = 12)$ and output the value for the next hour. The training and forecasting process we adopt is discussed in more detail in section 4.1. [Dorffner, 1996] presents a general discussions on NN architectures for time series prediction.

3. INTEGRATION

The basic idea of the integration of MAs and NNs – motivated in section 2.1– is that MAs carry NNs with them to the managed nodes to generate predictions and use them to manage nodes proactively. This results in a distributed system that employs mobile and intelligent components. First, we will discuss the

[1][Masters, 1993] presents NNs in a very exhausting, comprehensive, and practical manner.

potential conflicts that might occur in such a scenario. We will then derive a design which takes these conflicting properties into account.

3.1. Potential Conflicts

The heavy-weight character of NNs versus the need for light-weight MAs: The major preconception against NNs is that they are *heavy-weight.* However, it is important to distinguish several phases here: the design, the training and the application. As far as the first two phases are concerned, NNs can be called heavy-weight in a sense, since the construction and initial training of a NN generally is a time and data consuming task. But when this is done, they are actually a very compact representation of knowledge. A multi-layered feed-forward NN is characterized by the structural description (number of layers and number of neurons in every layer) and by the knowledge representation (the connection weights). The structural representation can be neglected because it consists of only a few bytes. The sample NN sketched in section 2.3 takes up less than 300 bytes. Thus, a MA with the typical size of a few **KB**[2] will not gain much *weight* by carrying it. Moreover, the resource consumption in the deployment phase is very low since calculating a forecast only takes a few simple mathematical operations.

The NN's inherent need for persistence versus the volatile nature of MAs: Training an NN, as we already stated, is a time and data consuming task. Therefore, the loss of the acquired knowledge has to be prevented by storing the trained NN persistently. How is this possible when NNs are used *and trained* by highly volatile MAs? Most existing persistence concepts for MA systems are specialized to achieve short-term error recovery, but not long term persistent storage.

The customization of NNs to local environments versus the roaming of MAs: The majority of NN applications involve a large degree of customization to a *local environment*[3] which is done during the training. For example, system load patterns vary among the hosts of a network because these hosts are used by different users with different habits. When a MA with an NN migrates to a specific node and trains the NN with the local system load values, the NN will learn the load patterns specific to this node. If the MA takes this trained NN to another node it will probably fail to generate correct predictions since this node exhibits different load patterns.

[2]This approximation is based on experience gained with our prototype implementation.
[3]In NM an environment may be defined as one host or a small portion of the network.

3.2. The Design

While the first potential conflict turned out to be no problem, the second and third conflict seem to be hard to handle when MAs carry NNs. But a general question has to be asked here: *Is it necessary and sensible for a MA to carry his one and only NN at all times?* It may be for some special applications[4]. But for the majority of cases, a less dynamic solution is not only sufficient but also necessary, especially when we look at the customization problem. The basic idea of our design is that NNs are bound to hosts rather than to MAs at runtime. At first, this might seem to be a contradiction to our goal. A description of the basic scenario will clarify the issues.

In this scenario we adopt the notion of health agents as described in section 2.2. A health agent is responsible for monitoring, predicting, and proactively controlling a certain aspect of the health of a single node or a set of nodes. At its start time a mobile health agent is given the NN, that it needs to perform its task. The node or network segment that the MA is assigned to defines its *environment.* Upon arrival, the MA delivers its NN to a local *NN-service* dedicated to maintaining numerous NNs belonging to different MAs (Figure la). Through this deposition of the NN, the MA is relieved from the burden of maintaining the NN. It measures the data and delivers it in regular intervals to the NN-service which in turn takes care of training the NN with the data, and generating predictions on demand (Figure lb). If the MA migrates or terminates, the NN is swapped to persistent storage by the service for reuse later on (Figure lc). If the MA returns at a later point in time, it does not need to remember that it already deposited its NN. When it tries to resubmit its original NN to the NN-service, the service notices that it already has the NN belonging to this MA in its repository and continues using it instead of registering a new one (Figure ld). As the MA migrates to different hosts, it implicitly distributes its NN, After a while, several NN-services on different hosts hold separate copies of the NN which are customized for the respective host's environment. This rather simple design has the following properties:

The notion of NNs is implicitly split in two. The NN that is carried around by the MA can be viewed as an *NN-type*, whereas the NNs maintained by the NN-services are *NN-instances*. These instances share a common structure and ability to learn a specific problem but they possess individual identities and different internal states. This also defines a new binding scheme for the NNs: The NN-type is bound to the MA while the NN-instances are bound to the hosts. A MA that travels across several nodes does not use one NN, but it uses several

[4]Such applications involve the recognition of globally observable patterns, i.e. patterns that exhibit the same behaviour wherever the MA goes.

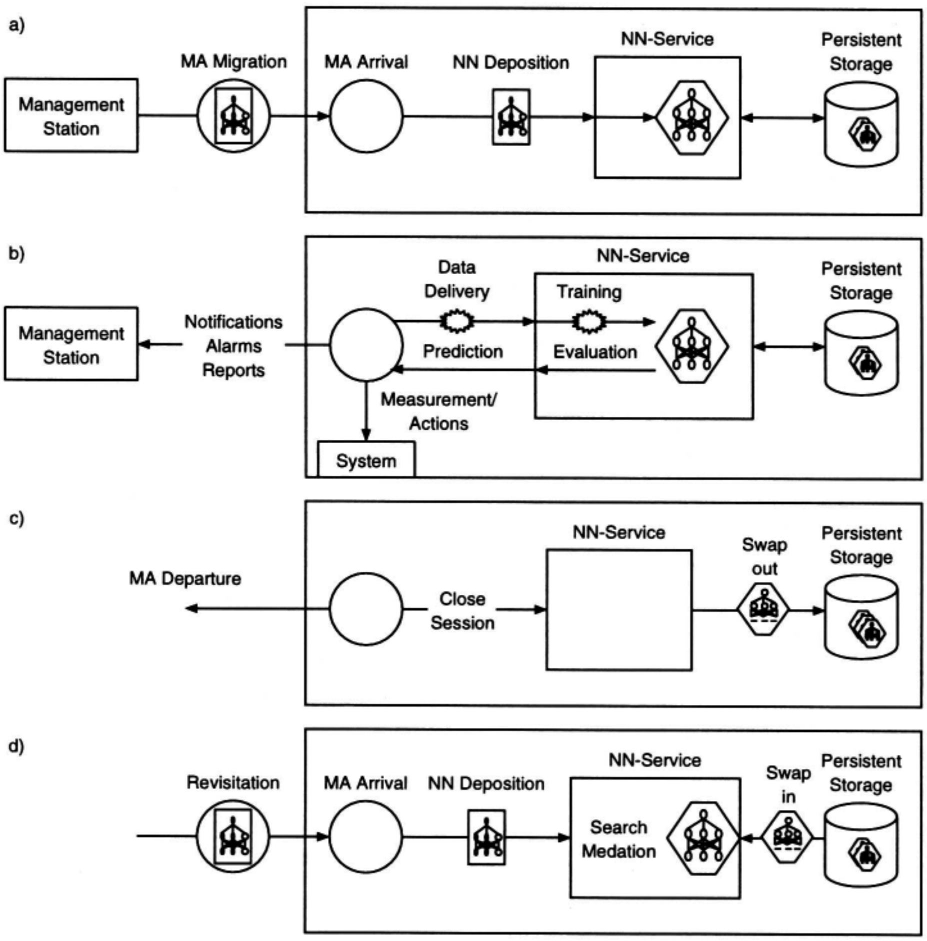

Figure 1. The basic scenario stages

customized instances of the same basic NN-type. This technique resolves the third conflict stated in section 3.1.

The MA does not have to take care of the NN's persistence and training. Therefore, the second conflict of section 3.1 is resolved.

The mechanism works transparently for the MA. On every visited host it executes the same operations: It registers its NN, requests predictions, and unregisters again before leaving. There is no need for the MA to hold a list of visited hosts in order to decide how to interact with a NN-service. Therefore, the deployment of the NNs is immune to the possible loss of an MA instance.

We achieve a decoupling of the *decision part* (the NN) and the *action-taking part* (the MA). The NN that decides what the future states of a system are, can be continuously trained and adapted to changing system characteristics. On the other hand, the MA whose code takes actions if the system state degrades, can easily be changed, removed or replaced without influencing the NN. This decoupling introduces a high degree of flexibility since the possible loss of a well-trained NN would make the replacement of a health agent very costly.

Summarizing these properties, we can conclude that the proposed design presents a good compromise between flexibility, persistence, and customization. It allows health agents to apply NNs in a decentralized way. At the same time highly mobile agents – let us call them *thin agents* – can be spawned by health agents and applied independently of the NN-usage to handle tasks autonomously that require such a high degree of mobility. All agents may communicate with each other to complete a common task cooperatively. Thus, we are able to fully exploit the advantages of MAs for NM. At the same time we can incorporate NNs as useful components in a decentralized NM system and achieve proactiveness through predictions. The issue of organizing NN training is discussed in the next section where we introduce a concrete prototype implementation of our design.

4. THE INTRAMANAGER SYSTEM

IntraManager is a scalable and flexible management system that decentralizes intelligence using MAs. It was designed to manage IP networks. However, we believe that the ideas proposed in our work can also be applied to other networks, like telephone or cellular phone networks since they rely on dedicated management platforms like TMN which could be enhanced. The IntraManager is build on the AMETAS agent system [Zapf et al., 1999]. IntraManager health agents can rely on several interfaces for measuring system parameters. These include e.g. SNMP agents and web server log files. A MA can use these data sources to calculate arbitrary health functions. Such a function may take several system parameters as input and evaluates to a single index characterizing the system state. This index is periodically calculated and compared with predefined thresholds to gauge the current state. If a threshold is exceeded for a certain amount of time, an action is invoked by the MA. The MA programmer can specify arbitrary actions for the different threshold events. The IntraManager system implements the design given in section 3.2. Thus, it is not only able to react to measured health indices but it can also predict them and derive proactive measures. The main motivation for the development of the IntraManager was the need for capacity planning within an enterprise network that has many low-bandwidth connections necessitating decentralized NM. Decisions have to be taken proactively concerning the extension of existing systems before the existing ones reach their limits. We are currently deploying the IntraManager prototype in such a testbed to investigate its applicability.

4.1. Automatic Training of Neural Networks

Our goal was to conduct proactive NM in a decentralized way to gain flexibility, scalability and robustness. Therefore, we gave a design that makes constant interventions by a central NM authority unnecessary and that is able to generate accurate predictions and take preventive actions. The framework requires the NN training to take place *automatically* at the managed nodes without central intervention. This poses a problem since the training process is generally the weak point of NNs. In practice, even a well-planned training process does not guarantee the success of a NN [Masters, 1993]. However, we assume a very specific application - the prediction of NM time series. Therefore, we can sooth this problem by introducing some constraints on the training process.

Helpful Constraints. The first constraint that is introduced by the nature of the problem is the fact that we do not have to deal with arbitrary functions. The time series observed in NM usually exhibit a superposition of several cyclic effects [Edwards et al., 1997]. Most NM parameters are directly or indirectly associated with human usage patterns which in turn obey to certain hourly, daily, weekly etc. cycles. Some of these cycles are known in advance which facilitates the construction of appropriate NNs. So these NNs can be constructed and tested centrally under human control and inserted into a central repository from which they are selected at the MA's start.

The second constraint is the choice of sample intervals in the magnitude of hours. That is, at the current stage we will not attempt to predict high-frequency data. Therefore, gaps in the measured data – which may appear when no measuring MA is present at the managed node – will only become critically large when the MA is absent for several hours. Gaps, i.e. missing values in the recorded time series, are a problem because this time series is to be used for the next training cycle of the NN. When the NN has already been trained it is possible to fill a gap in the time series by predicting the values in question. However, this can only be done when the gaps are not too large.

Another constraint can be applied concerning the NN *validation.* During the validation the trained NN is tested on a separate data set to ensure that it has good generalization capabilities. This task usually requires human supervision. However, the training and validation in the IntraManager context takes place *on-line*: A NN is trained with the collected data and validated during the deployment phase by comparing the predictions with the actually measured values. The difference of the two (*prediction error*) has a tendency to grow over time. When a certain error is exceeded, a new training run is initiated to accommodate the NN. Therefore, training and validation are ongoing processes [Edwards et al., 1997].

The Training and Prediction Process. For the actual training process the IntraManager uses the following scheme: In every *training epoch*, *N* random data samples – each consisting of an input window of length *n* and the $(n+1)$-th value as the target output – are presented to the NN. Subsequently the NN's weights are adjusted using the *conjugate gradient* method (see e.g. [Masters, 1993]). Several such *epochs* are conducted and the training run is stopped when the prediction error on the training data falls below some predefined threshold. We repeat this whole *training run* for a certain number of randomly initialized NNs and keep the NN which produces the lowest error. This NN is then deployed until the next training run is triggered (see section 4.1). Please note that the values of *N* and *n* are depending on the specific problem at hand and thus cannot be discussed here. Please refer to [Masters, 1993] and [Edwards et al., 1997] for a more general discussion on this topic.

The NNs used by the IntraManager have a single output neuron which enables them to predict *one* future value. To predict a longer window of values a *closed-loop forecast* can be conducted during which the predicted values are fed back into the NN as part of the input for the next forecast. The result is a small forecast time series enabling us to take proactive actions.

It should be noted that having a single output neuron does not really limit the power of the system. Having multiple output neurons for different time offsets usually renders the training much more difficult since it becomes harder for the training algorithm to minimize the prediction error. On the input side, one may actually use arbitrary data besides the pure time window (see section 2.3). Any kind of information that can be measured and that might be helpful for the training process can be used. One NN is dedicated to predict a specific system parameter or a compound index calculated from a number of parameters. To predict *k* different parameters one would use *k* NNs which can be operated by the same MA.

5. RELATED WORK

Several researchers have studied the application of NNs in NM. Kohonen self-organizing maps (SOM) are applied for event correlation in [Gardner and Harle, 1997]. A SOM is trained with alarms and learns to associate them with specific information on the root cause of the alarm. [Leray et al., 1996], [Biesterfeld et al., 1997], [Edwards et al., 1997], [Frank et al., 1999], and [Jobmann et al., 1997] present applications of feed-forward NNs similar to the one explained in section 2.3 for predictive and correlative purposes. The first major commercial application of NNs in proactive NM was recently introduced by Computer Associates. Their so-called *Neugents* are immobile agents using *functional link neural networks* to identify systems states and detect critical changes in their environment.

The application of NNs in combination with MAs to solve NM problems has not been addressed yet. An attempt to integrate MAs with an AI technique is made in [El-Darieby and Bieszczad, 1999] where the expert system JESS is used to correlate events within a MA system. However, they do not elaborate on the technical problems involved with this approach.

6. CONCLUSIONS

Our research integrates mobile agents and neural networks, two major technologies in contemporary distributed system management research. The goal of this integration is to enable MAs to use NNs for the prediction of future system behavior and to achieve decentralized proactiveness. We have designed and implemented an extension to our MA-based IntraManager system, that combines both technologies and shows that distributed system management can benefit substantially from this integration. Our future research will concentrate on the training process in specific application scenarios and its implications.

References

[Baldi et al., 1997] Baldi, M., Gai, S., and Picco, G. (1997). Exploiting Code Mobility in Decentralized and Flexible Network Management. In *Proceedings of the First International Workshop on Mobile Agents, MA'97, Lecture Notes in Computer Science Nr. 1219, Springer Verlag, 1997, ISBN: 3-540-62803-7,* Berlin, Germany.

[Biesterfeld et al., 1997] Biesterfeld, J., Ennigrou, E., and Jobmann, K. (1997). Neural Networks for Location Prediction in Mobile Networks. In *Proceedings of the International Workshop on Applications of Neural Networks to Telecommunications 1997 (IWANNT'97), Lawrence Erlbaum Associates,* Melbourne, Australia.

[Dorffner, 1996] Dorffner, G. (1996). Neural Networks for Time Series Processing. *Neural Network World,* 6(4):447–468.

[Edwards et al., 1997] Edwards, T., D.S.W.Tansley, Frank, R., and Davey, N. (1997). Traffic Trends Analysis Using Neural Networks. In *Proceedings of the International Workshop on Applications of Neural Networks to Telecommunications 1997 (IWANNT'97), Lawrence Erlbaum Associates,* pages 157-164, Melbourne, Australia.

[El-Darieby and Bieszczad, 1999] El-Darieby, M. and Bieszczad, A. (1999). Intelligent Mobile Agents: Towards Network Fault Management Automation. In *In Proceedings of the Sixth IFIP/IEEE International Symposium on Integrated Network Management*, pages 611–622, Boston/USA.

[Frank et al., 1999] Frank, R., Davey, N., and Hunt, S. (1999). Applications of Neural Networks to Telecommunications Systems. In *Proceedings of the European Congress on Intelligent Techniques and Soft Computing, EUFIT'99*, Aachen, Germany.

[Gardner and Harle, 1997] Gardner, R. D. and Harle, D. A. (1997). Alarm Correlation and Network Fault Resolution using Kohonen Self-Organising Map. In *Proceedings of IEEE Globecom'97, Vol.3*, pages 1398–1402, Phoenix, Arizona.

[Goldszmidt, 1993] Goldszmidt, G. (1993). On Distributed System Management. In *Proceedings of the Third IBM/CAS Conference*, Toronto, Canada.

[Jobmann et al., 1997] Jobmann, K., Tuchs, K.-D., Wietgrefe, H., Fröhlich, P., Nejdl, W., and Steinfeld, S. (1997). Using Neural Networks for Alarm Correlation in Cellular Phone Networks. In *Proceedings of the International Workshop on Applications of Neural Networks to Telecommunications 1997 (IWANNT'97), Lawrence Erlbaum Associates,* Melbourne, Australia.

[Kooijman, 1995] Kooijman, R. (1995). Divide and Conquer in Network Management using Event-Driven Network Area Agents, http://netman.cit.buffalo.edu/doc/papers/koo9505.ps, Technical Univerity of Delft, The Netherlands.

[Leray et al., 1996] Leray, P., Gallinari, P., and Didelet, E. (1996). A Neural Network Modular Architecture for Network Traffic Management. In *Proceedings of IEEE-CESA'96 IMACS multiconference, Symposium on Control, Optimization and Supervision, vol. 1 of* 2, pages 1091–1094.

[Levi and Schoenwaelder, 1999] Levi, D. and Schoenwaelder, J. (1999). Definitions of Managed Objects for the Delegation of Management Scripts (RFC 2592).

[Masters, 1993] Masters, T. (1993). *Practical Neural Network Recipes in C++.* Academic Press, ISBN 0-12-479040-2.

[Waldbusser, 2000] Waldbusser, S. (2000). Remote Network Monitoring Management Information Base (RFC 2819).

[Wooldridge and Jennings, 1995] Wooldridge, M. and Jennings, N. (1995). Intelligent Agents: Theory and Practice. *The Knowledge Engineering Review,* 10(2): 115–152.

[Zapf et al., 1999] Zapf, M., Herrmann, K., and Geihs, K. (1999). Decentralized SNMP Management with Mobile Agents. In *Proceedings of the International Symposium on Integrated Network Management (IM'99),* pages 623–635, Boston/USA.