Journal of Network and Systems Management, Vol. 9, No. 4, December 2001 (© 2001)

Decentralized Network Management Using Distributed Artificial Intelligence

Fernando Luiz Koch[1,3] and Carlos Becker Westphall[2]

Centralized approaches to Network Management have demonstrated a clear inadequacy for efficient management of large and heterogeneous computer networks. Considerable research is being carried out on decentralized approaches for network management. This paper presents the work on a practical application of Distributed Artificial Intelligence for computer network management. The objective is to implement a software platform using only Intelligent Autonomous Agents, integrated with the SNMP environment.

KEY WORDS: Distributed artificial intelligence; autonomous agents; distributed computing; knowledge management; computational cooperation; neural networks.

1. INTRODUCTION

The main task of Network Management system researchers is to develop new tools that work better than current ones for doing this laborious work. In this work we present our research about how to delegate as much work as possible to the machine, using Network Administrators as knowledge engineers who teach the machine how it should perform its work. The approach presented here is based on Intelligent Autonomous Agents.

Due to space limitations, we focus our presentation on functionality rather than technical details. We describe, in general terms, our approach to the problem and how it is implemented. Moreover, we present the framework used to build simple Network Management Agents and show how they would operate inside a Network Management Environment.

[1] Federal University of Santa Catarina, Network and Management Laboratory, Post-Graduate Program in Computer Science, Technological Center, P.O. Box 476, 88040-970, Florianopolis, SC, Brazil. E-mail: *koch@lrg.ufsc.br*
[2] Federal University of Santa Catarina, Network and Management Laboratory, P.O. Box 476, 88040-970, Florianopolis, SC, Brazil. E-mail: *westphal@lrg.ufsc.br*
[3] To whom correspondence should be addressed.

1.1. Advantages

There are quite a few predictable advantages in using Intelligent Autonomous Agents for any management system, Network Management being just one [1]. First, there is the "Intelligent" nature, which sounds promising. Having an Intelligent and adaptable system is usually better than having dedicated applications for specific solutions. Likewise, the word "Autonomous" gives us the idea of something that can work by itself or needs almost no human interference. Finally, "Agent" gives the impression of a "helper" or a wizard that somehow works between the machine and human.

Our objective is to project a system where the human System Administrators do not have to work as the main work force available. Instead, we want them to work as the truly intelligent—meaning cognitive—element in the management process, feeding the system with the rules of work—or knowledge—it needs to operate; no more boring work, but intelligent work!

The use of *Autonomous agents* is not an original idea and their application to Network Management has been explored before [2–4]. Our intention here is to show a practical approach and also a system design framework. Besides, we present some interesting concepts of distributed intelligence, independent learning, self-generation, interfacing through new upcoming standards—like CORBA—and adaptability.

2. DESIRED FEATURES—WHAT ARE WE PLANNING?

Our objective is to develop a decentralized computer network management platform borrowing existing concepts from distributed artificial intelligence. The guidelines for the system are: a high degree of adaptability, mobility, module reusability, self-generation and environmental plasticity [5].

Those guidelines sound simple but actually they hide what we have deep in our minds: "creating a system that can really work by itself, generating a new system for a new environment alone, developing new components for new circumstances, and learning as much as possible by itself asking us only when there is no other way."

From the guidelines of our project, we inferred SEVEN "commandments" that drove us towards what we could call "measurable" goals. This step is important for the sake of Project Management.

1. *Agents must be truly "autonomous":* We mean that any agent must be able to control itself and its own actions. It should be able to work independently from what happened to other agents and have enough freedom to adapt itself to any possible new environmental dynamics. Without this item we will have just a usual system where any new situation demands

full System Administration attention to properly react to new environmental conditions and set up new operating parameters.

2. *Agents should be "goal-driven" and "rule based":* A set of rules constitutes the knowledge base for any administration system—here, including human ones, and "goals," which are rules themselves, are the motivations that the agents have to work with. We define "static goals" as parameters set up during agent creation, and "dynamic goals," those added to agent behavior derived from environmental changes or incoming messages.
3. *The Agent environment must behave as a "global knowledge base"* where each data item or item of information stored by one agent is "sharable" throughout the whole system (with other agents). Agent intercommunication and knowledge exchange is essential to ensure that all needed information would be available for decision making. Besides, in the "global" system—meaning worldwide over the Internet—one agent could learn a new rule from another community and share that information within its own community, thus increasing the overall community's knowledge. We intend to reduce human interaction when we have one automatic system learning new "tricks" from another automatic system more knowledgeable than the first one in some field.
4. *Agents should be capable of learning new skills.* Either from other agents, the community of agents, the global community or, as a last resort, human beings. These new rules should be dynamically stored in Agent's local knowledge databases. The other possibility is inferring new rules based on environment data, along with current knowledge.
5. *Agents should be self-generating:* They should adapt themselves to new environments and generate sub-sets of their own characteristics creating new elements for new problems. This allows an agent to become a self-generating system, reducing human interaction as much as possible.
6. Specifically for Network Management Agents, *they should interface with commercial solutions using standard protocols.* This means that Agents should know, through rules in their databases, how to interact with well-known protocols such as SNMP (Simple Network Management Protocol) and CMIP (Common Management Information Protocol), how to retrieve information from log files, and also how to interact with other existing management systems. It is a marketing issue: making our system compatible means facilitating its acceptability by the market.
7. Finally, but no less important, Agents *should have a human-friendly interface.* The friendly interface between humans and machines is a natural-language-like protocol. Although very difficult to achieve, our work is being developed toward this final goal. In the mean time, we use a more formal specification for the human-agent communication lan-

guage and provide Application Program Interfaces (APIs) for developing other interfaces—such as HTML, NNTP and TELNET.

3. INTELLIGENT AUTONOMOUS AGENTS

In this work, we define “Autonomous Agent” as any software that assists people and acts on their behalf. How they are internally implemented or externally appear is up to the environment where they are inserted and their developers. It is not really important. These discrepancies between implementations can be clearly observed taking on different works about this subject like [2, 3, 6–9].

Therefore, *our idea* is that an *autonomous agent* is a software application that works by allowing human users to delegate tasks to these applications. Also, these applications should work independently, but on the other hand, should interact with human “masters” in order to acquire new knowledge and learn new “tricks.”

For example, in Network Management, Autonomous Agents could help in automating repetitive tasks, remembering rules that users frequently forget and summarizing complex data in understandable human-friendly reports or alerts.

3.1. Framework for a Generic Agent

The objective of the “Generic Agent” concept is to define the basic structures, mechanisms and abilities that guarantee a minimum standard behavior, internal and external, to any agent that is created. In other words, a Generic Agent is a template that the Agency System itself uses to create new agents. Over those templates new functionality can be added to make this new agent being created best fit into its destination environment [9].

The Generic Agent concept defines the following three modules, as shown on Fig. 1:

- *INFERENCE module*, which implements the Rule Deduction Engine and also stores the knowledge database. In future systems, it will hold the Artificial Neural Network part that is nowadays separated in customized Neural Agents.
- *INTERACTION module*, which performs the interface between the agent and the environment.
- *COMMUNICATION module*, which implements the message exchange procedures between agents. Communications are implemented by means of Knowledge Query and Manipulation Language (KQML) [10], which was chosen due to its strong commitment to agent applications.

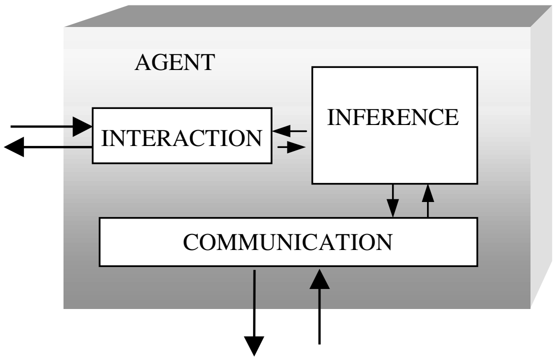


Fig. 1. Generic agent structure.

For extensibility purposes, the Generic Agent architecture is being adapted to be compatible with Common Object Request Broker Architecture (CORBA). We want our agents to interface with CORBA services, either making use of Distributed Objects or serving as CORBA repositories sharing internal functionalities.

4. NETWORK MANAGEMENT AND AUTONOMOUS AGENTS

Our objective is to show how Agents fit into a standard Network Management system. Our approach is to present a diagram from a regular system and then replace its internal components with another diagram, using Agent-based components with similar functionalities.

An ordinary Network Management could have a workflow like the one presented in Fig. 2. Data is collected from a managed device (1) using some "standard" management protocol, such as SNMP, CMIP or a proprietary protocol.

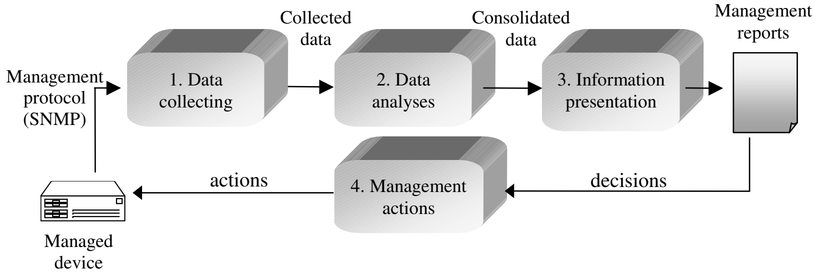


Fig. 2. Network management workflow.

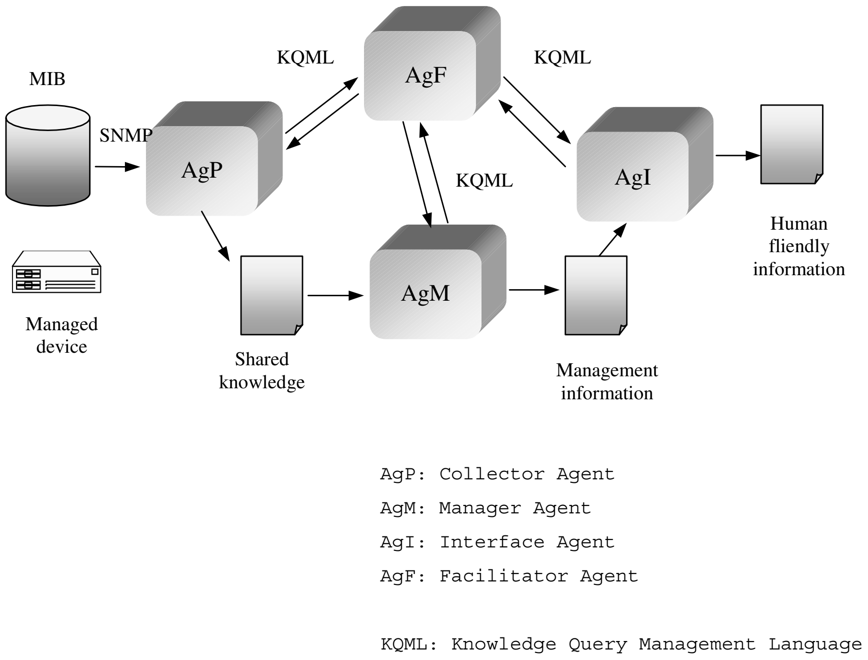


Fig. 3. Network management with agents.

After that, in step (2), the collected data is analyzed, and then condensed (3) creating the real management information. Later on, managers make their analyses (4) and, if required, take reactive actions in order to correct weak points of the system.

Based on this diagram, we have designed a new one, shown in Fig. 3, adapted to Agent technology. In this new structure we have three Management-oriented types of Agents: “Data Collectors” (AgP), “Managers” (AgM), and “Interfaces” (AgI). Moreover, belonging to Agent Community system architecture itself, we also have Communication Facilitators (AgF) used for coordination of knowledge exchange among Agents of the same community.

We remind the reader that every Agent in this design is developed based, ultimately, on the Generic Agent’s framework presented in the previous section.

4.1. Types of Autonomous Agents

Here we briefly introduce each Agent type and present some details about its implementation. We will not describe the details of coding here due to space limitation; furthermore, the code is in transition from C++ to Java. Therefore, we have focussed more on the general structure itself.

4.1.1. Data Collector Agents (AgP)

This type of agent retrieves data from devices and saves this information on local knowledge databases. This "knowledge" could be kept and processed locally and then "shared" with the whole community through the Agents' Communication module defined at Generic Agents framework, in Section 3.1.

The most important part of Data Collector Agent functionality—which is collecting data—is implemented by special knowledge rules loaded inside its knowledge database. These rules, in association with "native knowledge" implemented in the agent's code, allow them to interface managed equipment through SNMP, CMIP or proprietary protocols. The definitions of "rules," "native rules" and "knowledge base" will be explored in the next section.

A single Data Collector may collect information from several different devices. It could be set up with multiple goals, with one or more goals for each device.

4.1.2. Management Agents (AgM)

Management Agents (AgM) are responsible for the organization of operation between agents of the same community. In addition, they can implement a higher level of data analyses based on shared knowledge (data) coming from several different Collector Agents. And finally, we can imagine Management Agents coordinating multiple Data Collector Agents when working on a specific set of devices and later correlating that data.

Besides, we have other types of Management Agents associated with Agent Communities and not directly with Network Management environments. What we mean by this is that those agents will be present in any Agent Community, not just in those dedicated to Network Management.

These are, e.g., Agents that implement Agents Self-Creation, Agents that implement Neural Network structures, Wizard Agents—those that hold and share massive knowledge databases—and many others being projected in order to create an "Agent's Virtual World."

The types of Agents specially set up for Network Management purposes are:

- *Data Analyzers*, which analyze data derived from different knowledge bases, usually from distinct Data Collector Agents.
- *Data Consolidators*, which combine or correlate data coming from a set of distinct Data Collector Agents, creating new consolidated information.
- *Future Value Predictors*, which are implemented by using Neural Network structures designed for Time Series Prediction [11]. With this class

of Agents our intention is to implement entities that theoretically could predict future values in a time series. This feature is quite helpful in designing proactive management systems.

- *Value Classifiers*, which are also implemented by using Neural Network and permit data classification in scales, such as HIGH, NORMAL or LOW instead of linear values. This classification is implemented by Pattern Matching and is highly adjustable and it is even possible to create Neural Network structures that adjust themselves based on past values of the time series.

So far Neural Agents have been studied as an improvement in the system functionality. In a few words, Neural Agents work as "servers" for Neural Network structures, where batches of data are submitted for processing and results returned. In the future, we are planning to have "hybrid" Agents, with both rule-based and neural structures, inside the INFERENCE module (described in Section 3.1). The integration of these two technologies offers us indescribable possibilities that could be material for a forthcoming a paper.

4.1.3. Interface Agents (AgI)

Interface Agents (AgI) are the bridge between the Agent Communities and human managers. There are various possibilities for these interfaces, like e-mail interfaces, terminal-like interfaces and Web interfaces. Selecting which interface the human manager prefers to use depends on the kind of data output needed. The "friendlier" interface in some aspects are the terminal-like interfaces, where humans can "chat" with agents using a structured language similar to natural human language.

Interface Agents, like any other, have their functionality structured inside the Community and also the Agent World. When a new sentence is input, the local parser, which is implemented by means of rules inside the Knowledge database, will try to "translate" and understand it. If this is not possible locally, usually due to lack of local knowledge, the task will be escalated to a more skilled Agent—inside the community or not—which will try to get a translation for it. If in the end the Agents still cannot understand the human entered sentence, it will reply—as of current implementation—an "I can't understand your sentence" and ask to repeat.

4.1.4. Facilitator Agents (AgF)

Facilitator Agents (AgF) are communication management agents needed in an Agent community to work as brokers for the information exchange process. They also implement some useful tasks related to data exchanging like list of available service names, routing, mediation and translation.

5. IMPLEMENTATION

How are Agents implemented? This requires a lot of information, which cannot be included in a single article, but we provide some highlights.

Our motto on implementing the system was “keep things as easy as possible.” The brightest ideas are the simplest ones. Furthermore, things too complicated to implement will be even more complicated to maintain. Sometimes it is quite difficult to reconcile easiness with innovation, but we have tried our best. The second principle we used is that the system should not be process-intensive.

The *kernel part* for every agent in the system is the very same one defined in the Generic Agent structure proposal (Section 3.1). Thus we have only one single source code for every agent. The distinctions for each agent are made through the Rules loaded on its knowledge base and the native rules loaded on its code. Native rules follow the Java definition and are used strictly when the task is so specific that a Knowledge rule could not be implemented instead.

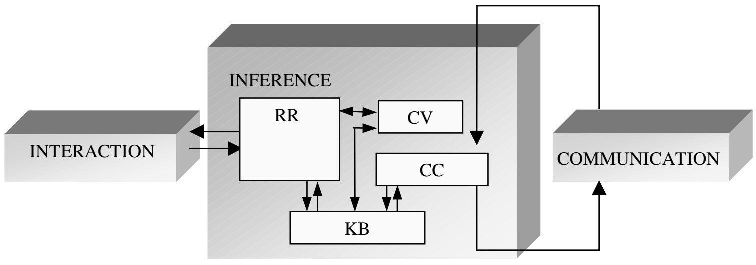
The *rules language*, called Mlog, is a subset of the Prolog language optimized for size and resource consumption reasons. Whoever develops in Prolog will not have any problems understanding Mlog. It is possible to “submit” knowledge to an Agent in the if-then-else rule format and the Agent will translate it to Mlog format—being helped by Interface Agents that have if-then-else-to-Mlog translation knowledge. A lot of Prolog programs will run on Mlog-based agents. We are designing Prolog convertors to allow “any” Prolog application to be compatible.

Programming in Mlog consists of: (1) Declaring some facts about objects and their relationships—for Network Management, we can see objects as devices, i.e.; (2) Defining some rules about objects and their relationships, and; (3) Asking questions about objects and their relationships. We will offer some examples later to make this clearer.

One major difference between Prolog and Mlog is that Mlog handles Exceptions like “unknown clauses” or “invalid values” instead of throwing out an error message. If an unknown clause is reached inside the rules resolution, rules are not analyzed when added to knowledge base (KB) but during execution—an *UnknownClause* exception is generated and the Cooperation Component (explained below) is activated. Thus, the Agent will try to “learn” that clause from the community before deciding to interrupt that execution sequence. The same applies to invalid values and other exceptions.

Once Mlog is learned, it can be easily understood how the *INFERENCE Module* is implemented. A diagram for this module is shown in Fig. 4.

The *knowledge base (KB)* is implemented by a set of Mlog sentences, like in Prolog. The *rules resolution component (RR)* is the Mlog engine and is based



RR: Rule Resolution Component

CV: Life cycle Component

CC: Cooperation Component

KB: Knowledge Base

Fig. 4. Detail for INFERENCE module.

on a full but reduced Prolog engine that processes the KB rules based on an expression being executed.

The *cooperation component (CC)* is a new feature, not usual to Prolog systems. It implements learning capabilities and is used when either new knowledge is required to accomplish a rule resolution, or new knowledge is offered to the Agent from the community. In other words, it deals with adding to or removing knowledge from internal knowledge bases and asking for new knowledge and later adding it to the community. A flowchart for CC vs. RR co-existence is presented in Fig. 5.

Finally, the *life cycle component (CV)* is an internal Mlog implemented component that compiles the set of "goals" and has an agent to pursue them in given moments of time. After that, it "submits" these goals, one by one, for execution in the rules resolution component.

The *INTERACTION module* is not a processing module *per se* but a set of rules and native methods declared inside an Agent to interface with environmental variables. In Network Management application, SNMP interface clauses are implemented in this module. The goal of creating an independent module for those rules and native rules is to better understand the system functionality. From the "compilation" point of view, the independent modules are linked inside the code depending, on the functions that will be exercised by Agent being built.

Finally, the *COMMUNICATION module* is much like the INTERACTION module: a set of rules and native methods targeting communication procedures. Things like TCP/IP package exchanging, KQML protocol formatting, commu-

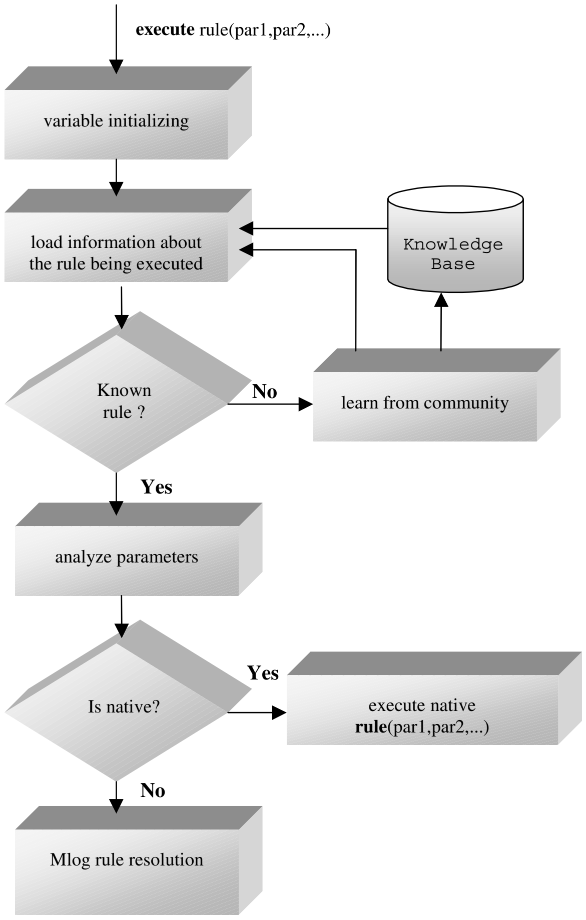


Fig. 5. Basic rule resolution flowchart.

nication session control, and others related to information exchange are treated in this module.

5.1. Practical Example—Data Collector Internal Knowledge Base

As an example, here we present an internal knowledge base for a very simple Data collector that retrieves and saves data from two different pieces of equipment and saves them in a local knowledge database.

```
INTERFACE module
SNMPget(IPAddress, OID, Value) :- native.
SNMPset(IPAddress,OID,Value) :- native.

COGNITIVE module
// *** Knowledge base (KB)
MIB("ifInOctets"," 1.3.6.1.2.1.2.2.1.10");
MIB("ifOutOctets"," 1.3.6.1.2.1.2.2.1.16");
Equipment("router","194.0.0.10").
Equipment("hub","194.0.0.11").
EquipmentMIBVar("router","ifInOctets",".0").
EquipmentMIBVar("hub"," ","ifInOctets",".0").
// *** Production rules (PR)
SetVars() :- setGlobal(DBName, $AgName+$Time+".db").
CollectDataEquipment(IPAddress, OID, Value) :-
       SNMPget(IPAddress, OID, ?Value).
SaveData(Time, EquipName, DescrMIB, Value) :-
       Assert("data",$Time, EquipName, DescrMIB, Value);
MonitorNetwork():-
       ?Equipment(?EquipName, ?IPAddress),
       ?EquipmentMIBvar(EquipName, ?MIBName, ?Interface),
       ?MIB(MIBName,?OID),
       ?CollectDataEquipment(IPAddress, OID+Interface, ?Value),
       StoreDataDB($DBName, Time, EquipName, MIBName, Value).

// *** Life cycle rules (CV)
On Start run SetVars().
Each 1min run MonitorNetwork().
Each 1hour run SaveDB($DBName).

// *** Cooperation Component (CC)
On UnknownRule($Rule) event run if(learn($Rule,30secs),resume,cut).
On InvalidValue event run cut.
On InternalError event if(gc(),resume,die).
```

* Of course, this is external representation and doesn't reflect internal knowledge storage structures inside Agents. We opted for a Prolog like output for sake of comprehensibility.

6. CONCLUSIONS

Autonomous Agents for Network Management, as well as for almost any other field in Computer Science, are still being discovered. The possibilities we have nowadays for applying Agent technology that are still unexplored are countless. Even though the implementation reported here has been successful in reaching some stated objectives, we still have a number of improvements and additional proposes. Our future works will be driven by the following motivations:

1. To create a series of new Agent modules and module plug-ins, in order to make them available to an increasing number of new working environments.
2. To integrate even more closely the promising field of Neural Networks inside the current INFERENCE module, in a transparent and extensible way. So, furthermore, we will have truly Hybrid Agents that will be able to take advantage of the best of both worlds. For Network Management, we see a direct utilization by creating neural structures to predict temporal series dynamically. This would be used in proactive Network Management through future value prediction.
3. To create a Global Wizard Structure that is a set of Internet available Agents to whom local Agents will refer in case they need new information that is locally unknown. In these structures we will store as much information as possible for all different fields where Agents will be applied.

This is still a brand new technology and there is a lot of research and development to be done. We really believe in Agent technology and feel that it could be used for any type of application. We also believe that this technology is one of the most promising research fields in software engineering today.

REFERENCES

1. J. Ferber, *Multi-Agent Systems an Introduction to Distributed Artificial Intelligence*, pp. 4–8, Addison-Wesley, France, 1999.
2. M. M. Cheikhrouhou, P. Conti, K. Marcus, and J. Labetoulle, A software agent architecture for network management: Case-studies and experience gained, *Journal of Network and Systems Management*, special issue on *Intelligent Agents for Telecommunications Management*, Vol. 8, No. 3, Sept. 2000.
3. E. Vayias, J. Seldatos, J. Bigham, L. Cuthbert, and Z. Luo, Intelligent agents for ATM network control and resource management: Experiences and results from an implementation on a network testbed, *Journal of Network and Systems Management*, special issue on *Intelligent Agents for Telecommunications Management*, Vol. 8, No. 3, Sept. 2000.
4. M. Kahani and P. H. W. Beadle, Decentralized approaches for network management, *Computer Communications Review*, ACM-SIGCOMM, Vol. 27, No. 3, pp. 36–47, 1997.
5. B. Burmeister, Cooperative problem-solving guided by intentions and perception, lectures notes

in distributed artificial intelligence, *Proc. of 11th Brazilian Symposium on Artificial Intelligence*, SBIA'92, pp. 77–91, Brazil, 1992.

6. P. Maes, Modeling adaptive autonomous agents. In C. Langton (ed.), *Artificial Life Journal*, MIT Press, Vol. 1, Nos. 1 & 2, New York, pp. 199–220, 1994.
7. H. A. Kautz, Bottom-up design of software agents, *Communications of the ACM* Vol. 37, No. 7, 1994.
8. P. Maes, Agents that reduce work and information overload, *Communications of the ACM*, Vol. 37, No. 7, 1994.
9. E. E. Scalabrini, L. Vandenberghe, H. de Azevedo, and J.-P. A. Barthés, Generic model of cognitive agent develop open systems, lecture notes in artificial intelligence, *Proc. of 13th Brazilian Symposium on Artificial Intelligence*, SBIA'96, pp. 61–70, Brazil, 1996.
10. Y. Labrou and T. Finin, A proposal for a new KQML specifications, TR CS-97-03, February 1997, Computer Science and Electrical Engineering Department, University of Maryland Baltimore County, Baltimore, Maryland.
11. A. Cichockil, Architectures and Electronic Implementation of Neural Network Models—Neural Networks for Optimization and Signal Processing, John Wiley, United Kingdom.

Fernando Luiz Koch obtained a degree in Computer Science in 1993 and M. Sc. in 1997, both at the Federal University of Santa Catarina. Currently, he is an associated researcher in the Network and Management Laboratory at the Federal University of Santa Catarina. His areas of interest are artificial intelligence, distributed systems and the implementation of autonomous agents for practical applications.

Carlos Becker Westphall obtained a degree in Electrical Engineering in 1985 and M. Sc. in 1988, both at the Federal University of Rio Grande do Sul, Brazil, and a Dr. Degree in Computer Science (Network Management) at the Université Paul Sabatier, France, in 1991. Presently, he is a Professor in the Department of Computer Science at the Federal University of Santa Catarina, Brazil, where he is the head of the Network and Management Laboratory.