CONEMAF: A MODULAR MULTI AGENT FRAMEWORK FOR AUTONOMIC NETWORK MANAGEMENT

Julien Boite, Gérard Nguengang
Advanced Information Technology Lab, Thales Communications SA, 160 Bd de Valmy, 92704 Colombes, France

Maurice Israël, Vania Conan
Advanced Information Technology Lab, Thales Communications SA, 160 Bd de Valmy, 92704 Colombes, France

Keywords: Autonomic network management, Cognitive agents, Situated knowledge, OSGi, Jess, OWL.

Abstract: Communication networks are growing both in terms of size and complexity. Because of the huge amount of monitored data to analyse and correlate, the management task that relies mostly on human operators is becoming time-consuming, labour-intensive and costly. The centralized management paradigm adopted by current management systems is no longer suitable for such networks. A distributed management with more autonomy delegated to network devices is therefore paramount to master this complexity efficiently.Multi-agent systems characteristics fit the requirements that must be taken into account to integrate autonomicity in networks. This paper presents CONEMAF, a novel modular multi-agent platform for autonomic network management. CONEMAF is fully distributed, allows situated knowledge analysis and implements a cognitive cycle for autonomic management. Implemented on top of OSGi, the present release makes use of well-proven Java-based COTS. CONEMAF has been deployed on Linux routers and is demonstrated for autonomic routing in a wireless mesh network.

1 INTRODUCTION

The management systems of future networks are expected to embed autonomic capabilities (Strassner J. K., 2006) in order to face the increasing complexity of communication networks (Ganek, 2003). This autonomic enablement implies that networks are able to self-manage (e.g., self-configure, self-heal, self-optimize, self-protect) (Horn, 2001). Their behaviour is set from high-level policies that they implement by dynamically adapting network parameters (routing, protocols, queue sizes, radio frequencies…) to the varying network conditions (changes in topology, resource availability, traffic demands…).

Designing an autonomic system for networks management involves several technologies and disciplines (Kephart, 2003) and has received significant research effort in the past five years. The generic architecture for autonomic information systems management proposed by IBM (IBM, 2006) is a common base for various works. FOCALE (Strassner J. A., 2006) adds semantics to this architecture. Algorithmic issues including learning capabilities (Tesauro, 2007), event correlation (Sterritt, 2002), data mining (Garofalakis, 2001), or bio-inspired mechanisms (Agoulmine, 2006) are under investigation to provide the expected autonomic capabilities.

On the implementation side, attempts are made to add more flexibility in network protocol design: (Keller, 2008) proposes a framework for self-composing protocol stacks in multiple compartments to replace the existing non-flexible Internet layers, and (Manzalini, 2006) provides a platform for autonomic service composition.

Focused on the middleware part of the framework, another research trend consists in integrating mobile agents in the network to collect distributed information or perform remote computations (Lefebvre, 2005). A combination of both mobile and stationary agents to manage mobile ones is proposed in (Ray, 2008).

Starting from the network administrator's viewpoint, our aim is to provide a software platform

Boite J., Nguengang G., Israël M. and Conan V. (2010).
CONEMAF: A MODULAR MULTI AGENT FRAMEWORK FOR AUTONOMIC NETWORK MANAGEMENT.
In *Proceedings of the 2nd International Conference on Agents and Artificial Intelligence - Agents*, pages 224-231
DOI: 10.5220/0002735502240231

that supports autonomic self-management of the network governed by high-level policies set by the operator. Multi-agent architecture principles support coordination of distributed intelligent components, which match our requirements. The most popular multi-agent implementations, such as JADE, Zeus or Cougaar (Ricordel, 2000) (Fang, 2005) follow the FIPA specifications for interoperability purposes (FIPA, 2005). These efforts, and Jade in particular, have guided some of the choices we have made. However, these implementations do not fulfil one of the main requirements for a scalable and robust network management system. Indeed, they rely on a centralized component (the FIPA Agent Management System) that does not meet the full distribution characteristic we need.

The major contribution of this work is the specification and design of CONEMAF (COgnitive NEtwork MAnagement Framework), a flexible and fully distributed platform that facilitates the implementation and the dynamic deployment of autonomic control loops in network elements. CONEMAF is able to scale to large networks in a robust and fault-tolerant way. It can be deployed on a large variety of network equipments (routers, home-gateways, or even terminals). A first release has been implemented and deployed on Linux routers to demonstrate enhanced routing features in a wireless mesh network.

The paper is organized as follows: section two describes the autonomic networking challenges in terms of architecture and software requirements. Section three introduces the modular CONEMAF platform, the baseline technologies used, and details its components. Section four demonstrates the use of CONEMAF to support autonomic congestion management in a mesh environment. Section five concludes with future works.

2 THE AUTONOMIC NETWORKING CHALLENGE

Networks are complex and costly systems that cannot be changed overnight. Enhanced management capabilities of converged future networks must be introduced gradually. The corresponding architecture and implementations must thus address the conflicting requirements of providing new management features and of deploying on legacy networks.

2.1 Autonomic Network Architecture

In the autonomic network vision, each network equipment (be it a router, a gateway…) is potentially considered as an autonomic element. An autonomic element is capable of monitoring the network state and modifying it based on conditions that administrators have specified. This "cognitive cycle"consists in constantly monitoring the network state, making decisions and executing reconfiguration actions.

To support gradual introduction of autonomic capabilities, the cognitive cycle is implemented in software. In the remainder of the paper, this software add-on embedded in network equipments and executing the cognitive cycle is referred to as the "cognitive network manager" (CNM).

Network equipments are physically distributed and the configuration of a given equipment often interplays with that of other equipment configurations. So, the representation of the network context that every CNM makes cannot be limited to its local point of view. Using a centralized approach to provide this complete picture is not feasible as it would not scale and be tolerant to faults. This means that communication between distributed autonomic elements is required to exchange relevant information as input for the decision making process.

Broadcasting messages throughout the entire network is not an option and the autonomic network management system must limit the communications to a defined range. Considering that knowledge about an entire network is not necessary to make fast local decisions either, information exchange is limited to compartments (subsets of neighbouring network elements). As a consequence, each CNM has a situated knowledge corresponding to its compartment view. Decisions are made locally, based on information gathered and exchanged between neighbouring entities in the defined compartment view.

Because of the variety of capabilities of network elements, some CNMs may be more complex than others. These particular CNMs could deal with higher-level knowledge collected by a set of CNMs in the specified domain they manage. With this knowledge, possibly complemented by exchanges with other domain managers, they could make global decisions solving problems that standard CNMs cannot address locally, control and adjust policies to fit the desired system behaviour, and learn from previous experience (Figure 1).

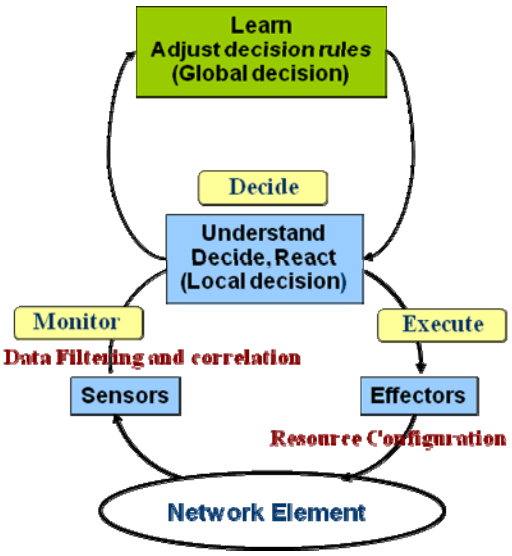

Figure 1: Multi-layered cognition and decision capabilities.

The autonomic network management architecture must thus support flexible and dynamic compartment definition.

2.2 Software Requirements

Implementing the architecture described above leads naturally towards multi-agent platforms since they aim to include communication, perception and action functions (Ferber, 1995). Each CNM is embedded in a network element and the autonomic network management framework should provide it with the following functionalities:

- network topology discovery
- neighbourhood discovery
- semantic communication
- network element control, via services which are declined to sensors and effectors
- scheduling (in order to plan behaviours execution).

The multi-agent implementation must satisfy further requirements. First of all, the autonomic network management system needs to be fully distributed in order to scale up to large or constrained networks and to be fault tolerant. Then, considering that CNMs are embedded in heterogeneous hardware devices with sometimes hard constraints implies that the solution must be lightweight and easily adaptable.

Moreover, the system must support reasoning capabilities to make fastdecisions in order to deal with applications or services that are not tolerant to delays. This means they must manipulate knowledge (situation, operator policies). The distribution of entities implies that knowledge is distributed. As a consequence, the knowledge representation must be uniform in order for entities to exchange understandable information.

Finally, the system must provide communication functions for entities to cooperate and collaborate. Modular sensors and effectors that cognitive network managers will use to gather data and configure equipments are also necessary.

3 CONEMAF PLATFORM

The CONEMAF platform has been designed to address the software requirements previously stated. It is composed of robust baseline COTS, which, once combined in the CONEMAF framework, provide the CNMs with the expected capabilities. Developed in Java, it can be easily integrated in various network equipments running a JVM (routers, home gateways, end devices) and allows fully distributed autonomic management of different types of networks such as wired or wireless.

3.1 Baseline Technologies

3.1.1 OSGi

Because CNMs aim at being embedded in network devices where the computational resources are devoted to routing, they should be designed in such a way that the required modules are only activated depending on devices capabilities. OSGi, for Open Services Gateway initiative (OSGi Alliance, 2009), is a Java platform providing functionalities for service-oriented programming. It allows developing smaller independent modules that can dynamically provide services to other modules and consume those that other modules offer. In this framework, modules, called bundles, can be loaded, started, replaced or stopped at runtime. Moreover, this platform allows developing bundles with clear separation between interfaces and their implementation in a natural way. To summarize, OSGi makes programming service-oriented applications easier and provides high adaptation capabilities, which is a required property for integrating autonomic elements in future network equipments.

Other assets of the OSGi platform are the ease of bundle version management and, above all, the plugin deployment facilities. In fact, launching the OSGi framework is straightforward: some tools like

PaxRunner (PaxRunner, 2009) ease the management of bundle provision inside interchangeable OSGi framework implementations. Managing bundles life cycle (install, run, stop, uninstall) is also simple.

Another important benefit of the OSGi technology is that Cisco network equipments already integrate an implementation of the OSGi platform (Cisco, 2008). Remote deployment of OSGi bundles is the only thing to do to integrate the autonomic network management system in a Cisco piece of equipment.

3.1.2 Web Ontology Language and Semantic Web Rule Language

A cognitive network manager needs to have a "world" representation and capture the technical know-how in order to work autonomously with respect to the operator objectives. Therefore, a tool that allows the network expert to express its knowledge in a machine-readable way is required.

An ontology is a « shared and common understanding of a domain that can be communicated between people and heterogeneous and distributed systems » (Fensel, 2001). It enables representing domain background knowledge in a machine understandable form (Studer, 1998). Using such a formal model within our distributed architecture is appropriate. An ontology defines a set of concepts, properties, relationships, constraints and axioms that provide rules that govern them.

The Web Ontology Language (OWL) (W3C, 2004) is the W3C standard for ontological modelling. It has been designed to provide a common way to process the content of web information. The OWL standard defines three increasingly expressive dialects: OWL Lite, OWL DL and OWL Full. OWL Full contains all the OWL language constructs but has no computation guarantees because it introduces too many possibilities. OWL DL is a sublanguage of OWL Full and relies mostly on description logics (DL). OWL DL is computationally decidable and more appropriate for knowledge representation when inference is needed. OWL Lite is a subset of OWL DL and suits well for expressing basic classification hierarchy and simple constraints. Although originally defined as an important part of the semantic Web suite, OWL is emerging as the major standard for knowledge representation.

However, OWL constructs do not allow the formalization of rules on top of the ontology. Among many proposals aiming at enhancing OWL knowledge bases with rules, the Semantic Web Rule Language (SWRL) (W3C, 2004) is probably the best known and most established. SWRL provides the means to define rules that extends the OWL set of axioms.

3.1.3 Jess Inference Engine

Cognitive network managers need reasoning capabilities to make decisions according to policies defined by network administrators. An inference engine performs reasoning from declarative facts. Jess, for Java Expert System Shell (Friedman-Hill, 2003), is a fast and powerful rule engine for the Java platform, which supports development of rule-based systems that can be tightly coupled to code entirely written in Java. Jess has been integrated with several agent frameworks and other tools like the popular ontology editor Protégé (Protégé, 2009). Jess, which supports both forward and backward chaining, has been integrated in the CONEMAF platform to provide such reasoning capabilities.

3.2 Modular Framework

CONEMAF is built on top of OSGi and follows the modular principles that OSGi enables. Software upgrade, deployment over heterogeneous network elements are thus facilitated.

3.2.1 Framework Components Overview

Figure 2 represents the different components the CONEMAF platform is made of from a software point of view. The core framework is composed of components that are essential for the cognitive network manager to play its role. This includes a scheduler to trigger the execution of behaviours, an inference engine for decision-making, and a blackboard, which acts as an organized common space for information sharing. Topology, discovery and communication services are also implemented as modules. The main benefit of the framework resides in the simplicity of adding, deleting or changing one of its components. Behaviours and network element controllers that may be adapted to the type of device they are embedded in particularly aim at exploiting such a benefit. All these components are individually described in the following section.

3.2.2 Modules Description

Each component of the cognitive network manager is implemented as a module, called bundle in the OSGi terminology. The present release of

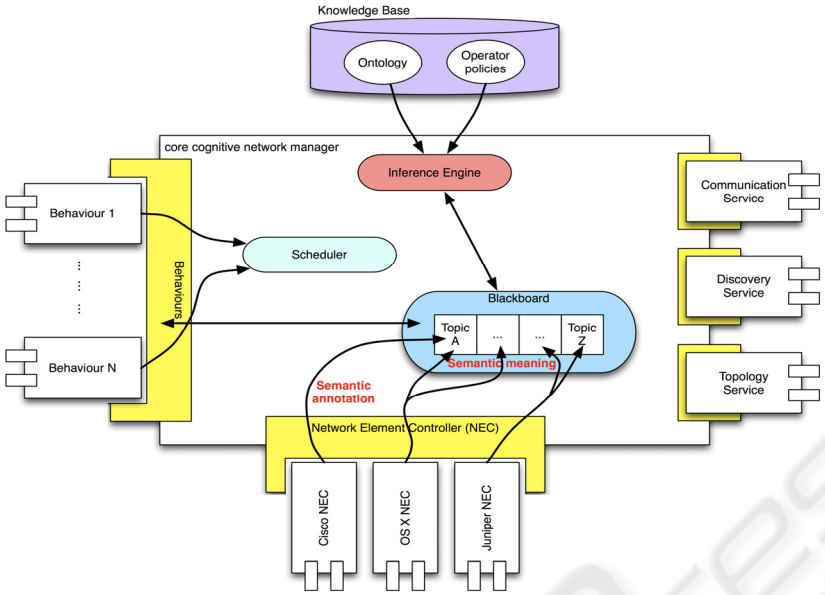

Figure 2: CONEMAF components.

CONEMAF includes a first implementationof the following modules:

- Topology service: based on a regular exchange of hello messages, constructs, updates and exchanges adjacency tables that reflects the physical network topology
- Discovery service: maintains information concerning neighbouring cognitive network managers (IP addresses, network interfaces used to reach them…)
- Communication service: fundamental service that provides communication functions with high-level semantics (communication acts, content identification)
- Network element controller: gathers network equipment status and publishes these different pieces of information (interfaces, routing…) in the blackboard. It also provides means to enforce reconfigurations in the network equipment.
- Behaviours: a behaviour realizes tasks or computations that are inherent to a specific objective. Two kinds of behaviours have been developed, differentiated on how often they are executed:
  - one shot: is executed only once
  - periodic: is executed at regular intervals.

  The execution of behaviours is triggered by the scheduler. Their life cycle, depicted in Figure 3, can be controlled.
- Scheduler: schedules the execution of schedulable components registered beside it (behaviours basically)

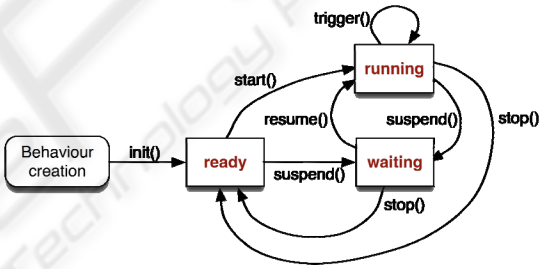

Figure 3: Behaviour life cycle.

- Blackboard: provides writing and reading functions in a shared space. We have extended LighTS (Picco G. P., 2005) (Picco G. P., 2001), a light Java implementation of the Linda model of tuple spaces (Wells, 2003). The blackboard contains different topics semantically linked to concepts of the ontology. Each topic is implemented as a tuple space
- Inference engine: in charge of making high-level decisions, it can be considered as the brain of a cognitive network manager. The ontology and the operator policies are loaded inside it, so as declarative facts representing the situation. Regarding to defined policies, decisions can be made as rules are triggered. The inference engine makes decisions that may need additional information for reconfiguring the network equipment and delegates these kind of specific processes to behaviours.

4 AUTONOMIC CONGESTION MANAGEMENT IN A DYNAMIC ENVIRONMENT

The purpose of this experiment is to offer an initial validationof the applicability of CONEMAF to perform autonomic network management. The objective of using autonomic management is to provide faster adaptation of the network to varying load conditions. This experiment aims to highlight the use of the framework to overcome congestion problems in wireless networks, where bandwidth often fluctuates. In this specific case, CONEMAF provides proactive route selection on top of the legacy routing protocol. The experiment has been carried out with a virtualization tool that emulates network equipments and links. The same scenario is run without and with CONEMAF and performances are compared in both cases.

4.1 Initial Situation

Let's consider the mesh network in Figure 4. It is composed of five routers and the scenario also involves two end devices. OSPF (Moy, 1998), one of the standard Internet routing protocol, is executed in this mesh network. A video flow is running across the network from a server to a client. In this context, the scenario consists in evaluating the visual impact of different failures or disruptions without and with CONEMAF cognitive network managers.

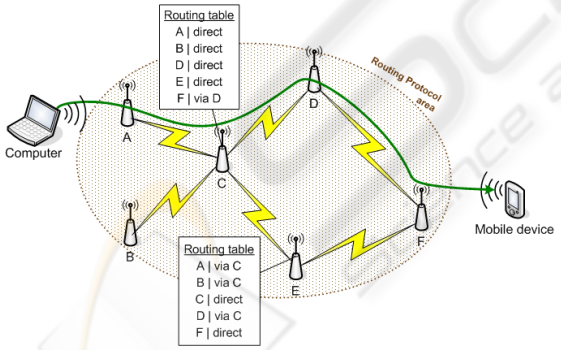

Figure 4: Initial situation.

4.2 Without CONEMAF

First of all, without cognitive network managers, when a link fails on the path from source to destination, the routing protocol is able to detect the failure and converges back. The video flow is affected (stopped) and its diffusion restarts when the routing protocol has converged (this can be quick).

Then, still without cognitive managers, limiting bandwidth capacity on a link between source and destination simulates a link disruption. This is the most interesting situation because, as there is no link failure, the routing protocol cannot do anything (it is not able to detect this disruption). The quality of the video flow remains affected (e.g., blocks still appear).

4.3 CONEMAF in Action

Cognitive network managers aim at anticipating corrective actions to problems that can arise in a network. This first demonstration using the framework consists in detecting a link disruption and reacting to this problem as soon as it occurs by redirecting the video flow.

To achieve this, a policy defining a threshold for available bandwidth on network interfaces is included as a rule in the inference engine. It looks like this:

```
(interface ?name … ?availableBW …)
^ (?availableBW <= threshold)
=> (changeRoute ?name)
```

The network element controller regularly gathers and publishes information on interfaces (name, state, available bandwidth…) in the blackboard. These factual pieces of information concerning every interface are retrieved by the inference engine and inserted into it under that form:

```
(interface <name> <state>
<availableBW> <otherParameters>)
```

Simple behaviours realize the following actions:

- exchange local routing table with direct neighbours (Figure 5)
- use received neighbouring routing tables to compute alternative paths (the algorithm is illustrated in Figure 7) to reach each destination contained in the local routing table (Figure 6).

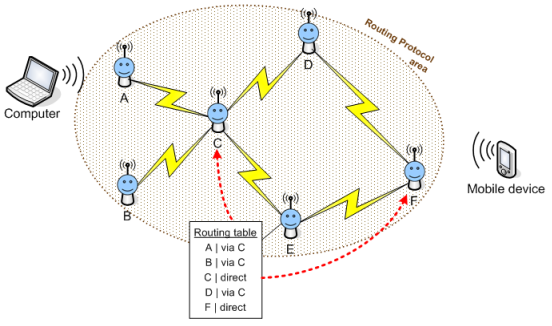

Figure 5: Local routing table exchange between neighbours.

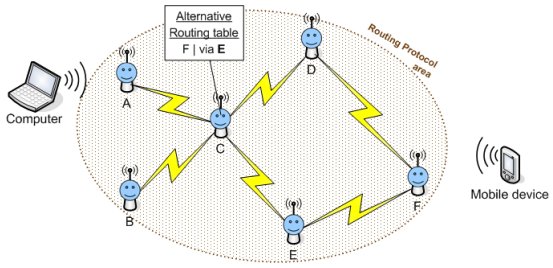

Figure 6: Alternative routing table computing.

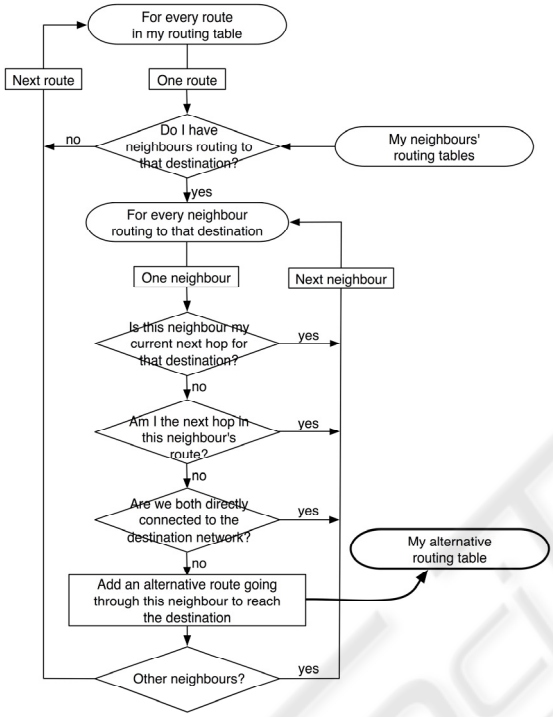

Figure 7: Alternative routing table computation algorithm.

When a disruption occurs on a given link and the defined threshold is reached, the rule inserted in the inference engine is triggered and a fact is published in the blackboard topic associated to orders. The network element controller then reads this order for changing the route(s) using the relevant interface (whose name is given in the inference result). It makes use of alternative routes computed by the behaviour in charge of this task to select one to be applied. If existing, this alternative route is enforced in the network equipment routing table and the flow is instantaneously redirected (Figure 8).

Enforcing an alternative route is a solution to face the disruption. Other solutions could be envisaged and the cognitive network manager would have to choose the most appropriate according to the situation. This simple reaction gives quite good results: even if we can see small pixels for less than one second, the video remains fluent.

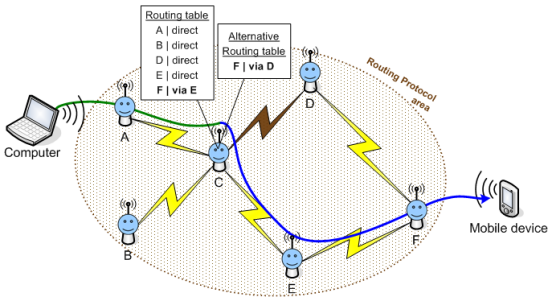

Figure 8: Facing the disruption by redirecting the video flow.

5 CONCLUSIONS AND FUTURE WORKS

This paper has presented CONEMAF, a platform that implements Cognitive Network Managers in Java. CONEMAF emphasizes autonomous distributed cognitive network managers that are able to monitor and act on the network equipment they are embedded in. They share an ontology that allows them to understand information they gather and exchange. Following high-level policies defined by operators, they can reason, make decisions and cooperate autonomously.

Based on the OSGi platform, CONEMAF is made up of easily deployable components and provides the modularity required to manage heterogeneous equipments in future converged networks. The first release has demonstrated autonomic routing capabilities in a simple use case.

Further work on the ontology and the inference engine is the next step towards extended autonomic capabilities. While still paying a particular attention to the framework modularity, further developments will then focus on giving the cognitive network managers learning capabilities to evaluate and adjust decisions based on their past experiences.

REFERENCES

Agoulmine, N. B. (2006). Challenges for Autonomic Network Management. *First Conference on Modelling Autonomic Communication Environment (MACE'06)*.

Cisco. (2008). *Cisco OSGi Add-On as Universal Middleware for Your Applications*. Retrieved from http://www.cisco.com/en/US/prod/collateral/routers/ps9701/data_sheet_c02_459075.html

Fang, F. R. (2005). *Technology Review of Multi-Agent Systems and Tools*. Navy Personnel Research Studies and Technology.

Fensel, D. (2001). *Ontologies: Silver Bullet for Knowledge Management and Electronic Commerce.* Springer Verlag.

Ferber, J. (1995). *Les systèmes multi-agents : Vers une intelligence collective.* Paris, France: InterEditions.

FIPA. (2005). *FIPA Specifications*. Retrieved from Foundation for Intelligent Physical Agents: http://www.fipa.org/specifications/index.html

Friedman-Hill, E. (2003). *Jess in action : rule-based systems in Java.* Greenwich, CT, USA: Manning Publications Co.

Ganek, A. G. (2003). The dawning of the autonomic computing era. *IBM Systems Journal*.

Garofalakis, M. R. (2001). Data Mining Meets Network Management: The NEMESIS Project. *Proceedings of DMKD'2001 http://www.dataspaceweb.net*.

Horn, P. (2001). Autonomic computing: IBM's perspective on the state of Information Technology.

IBM. (2006). An architectural blueprint for autonomic computing. *White paper*.

Keller, A. H. (2008). A System Architecture for Evolving Protocol Stacks. *Computer Communications and Networks, 2008. ICCCN '08 (Invited paper)*, 1-7.

Kephart, J. C. (2003). The Vision of Autonomic Computing. *IEEE Computer, 36*, 41-50.

Lefebvre, J. C. (2005). A Network Management Framework Using Mobile Agents. *Journal of Computer Science*.

Manzalini, A. Z. (2006). Towards Autonomic and Situation-Aware Communication Services: the CASCADAS Vision. *Proceedings of the IEEE Workshop on Distributed Intelligent Systems: Collective Intelligence and Its Applications (DIS'06)*, 383-388.

Moy, J. (1998). *RFC 2328 - OSPF Version 2*. Retrieved from www.ietf.org/rfc: http://www.ietf.org/rfc/rfc2328.txt

OSGi Alliance. (2009). *OSGi - The Dynamic Module System for Java*. Retrieved from http://www.osgi.org

PaxRunner. (2009). *Pax Runner*. Retrieved from http://paxrunner.ops4j.org

Picco, G. P. (2001). *LighTS*. Retrieved from LighTS: http://lights.sourceforge.net

Picco, G. P. (2005). LighTS: a lightweight, customizable tuple space supporting context-aware applications. *Proceedings of the 20th ACM symposium on Applied computing*, 413-419.

Protégé. (2009). *Protégé*. Retrieved from Protégé Home Page: http://protege.stanford.edu

Ray, P. P. (2008). Distributed Autonomic Management: An Approach and Experiment towards Managing Service-Centric Networks. *International Conference on Service Systems and Service Management, 2008*, 1-6.

Ricordel, P.-M. D. (2000). From Analysis to Deployment: A Multi-agent Platform Survey. *Proceedings of the First International Workshop on Engineering Societies in the Agent World (ESAW'00), 1972*, 93-105.

Sterritt, R. B. (2002). Towards Autonomic Computing: Effective Event Management. *Proceedings of the 27th Annual NASA Goddard/IEEE Software Engineering Workshop (SEW-27'02)*, 40-47.

Strassner, J. A. (2006). FOCALE - A novel autonomic networking architecture. *Latin American Autonomic Computing Symposium (LAACS)*.

Strassner, J. K. (2006). Autonomic Systems and Networks: Theory and Practice. *10th IEEE/IFIP Network Operations and Management Symposium, 2006. NOMS 2006*, 588-588.

Studer, R. B. (1998). Knowledge Engineering: Principles and Methods. *Data Knowledge Engineering, 25*, 161-197.

Tesauro, G. (2007). Reinforcement Learning in Autonomic Computing: A Manifesto and Case Studies. *IEEE Internet Computing*, 22-30.

W3C. (2004). *OWL Web Ontology Language Overview*. Retrieved from http://www.w3.org/TR/2004/REC-owl-features-20040210/#s1.3

W3C. (2004). *SWRL: A Semantic Web Rule Language Combining OWL and RuleML*. Retrieved from http://www.w3.org/Submission/SWRL

Wells, G. C. (2003). Linda implementations in Java for concurrent systems. *Concurrency and Computation: Practice & Experience*.