Horizon Project

ANR call for proposals number ANR-08-VERS-010
FINEP settlement number 1655/08

Intelligence-oriented tools for networking

WP1 - TASK 1.3: Intelligence-Oriented Tools for Networking

Institutions

GTA-COPPE/UFRJ
PUC-Rio
UNICAMP
Netcenter Inform´atica ltda
LIP6 Universit´e Pierre et Marie Curie
Telecom SudParis
Devoteam
Ginkgo Networks
VirtuOR

Contents

List of Figures

Chapter 1

Introduction

Autonomic networks constitute a fast expanding field. The control and management of thousand of equipments consists of many repetitive and heavy tasks. The network community is becoming increasingly aware of the necessity to replace the human administrator. In fact, it would be far more economical and more effective to let the equipment handle a part of the tasks assigned to the human administrator. In that case only, the administrator will have to define the main rules of the system operations. In such a context, the network can be self-configured, self-managed, etc. while respecting the policy decided by the operator.

Since the emergence of multi-agent systems, many researchers tried to apply this approach to telecommunication networks. In fact, this technology provides a lot of flexibility, adaptability and autonomy to the networking area. This scheme concerns mainly functions of network management such as fault management [49], detection of intrusions [21], etc.

The multi-agent approach aims to overcome the incapacity of the classic artificial intelligence (AI) to solve complex and distributed problems. Indeed, AI offers very powerful and usually optimal algorithmic schemes, but it is not able to provide solutions to problems such as the air traffic management, functioning of an electronic market, control and management of a communication network, etc. These very highly distributed systems cannot be managed in a centralized way because this would raise bottleneck problems and faults by the central control entity.

We propose the multi-agent technology as a tool to reach the autonomy in virtualization network control and management. This approach has been tested and validated through many experiments [32, 37]. In this deliverable, we describe the autonomic communication paradigm, its properties, and the need for a new architecture (section 2). Then, we focus on what we have to add to go from a classic network to an autonomic one (section 3). We present some related works on the combination of these two fields: the artificial intelligence, and distributed artificial intelligence and the telecommunication networks.

We present the properties of an agent and of a multi-agent system able to respond to our need to make the home network autonomic (section 4.1 and 4.2). Then, we analyze the different existing platforms (network oriented and agent oriented: section 4.2.

Chapter 2

Multi-agent solution

If the agents own a part of the capabilities to make the network autonomic, they have to be organized as a group (such as multi agent system) to get all the capabilities needed to make the network totally autonomic. Multi-agent systems have been used in many areas and have proven their utility. Thanks to their properties, the agents represent a good tool to make networks autonomic. Indeed, a multi-agent system consists of a set of agents which [20]: (1) are able to communicate, (2) possess their own resources, (3) perceive their environment (to a limited extent), (4) have a partial representation of their environment (and perhaps none at all), (5) have a behavior which aims at realizing their purposes. This realization is done by taking into account their resources and their capabilities, as well as their representation and the inward information according to their perception. Their main characteristics are developed in the following section.

2.1 Characteristics of the agents

Multi-agent systems can constitute a good tool to provide the autonomic scheme by guaranteeing the different characteristics which seem necessary to reach an autonomic behavior. We presented this autonomic behavior in sections 2 and 3. In the following, we demonstrate that all these characteristics are indeed offered by the multi-agent solution:

- ***Decentralization***: The multi-agent approach is decentralized by definition. No agent possesses a global vision of the system and the decisions are taken in a totally decentralized way. This decentralization aims at overcoming the incapacity of the classic Artificial Intelligence to operate in the current systems which are more and more distributed and decentralized (road traffic control, electronic market, communications network, etc.);

- ***Reactivity***: The agent acts and reacts to the environment in which it evolves and participates in the good functioning of the global system. One of the basic attributes of an agent is to be *situated* (situatedness, [6]). That is, an agent is a part of an environment and it makes its decisions according to what it perceives of this environment and to its current state. This basic characteristic is very important in a context of highly dynamic networks, in which the decisions have to suit best the current conditions. With an agent-based tool, we have this attribute in a secure way;

- ***Proactivity***: An agent can be endowed with more elaborate capabilities than the simple fact of reacting to its environment. The agent is capable of settling goals and realizing them by implementing plans, starting cooperation with other agents, etc. In this case, the agent has more knowledge on his capabilities and on those of the other agents and is able to set up a strategy allowing it to evolve in its environment and to reach its objectives;

- ***Sociability:*** One of the interesting features of the multi-agent approach is its ability to distribute the intelligence between the different agents composing the system. This implies that an agent can handle some tasks individually but cannot make everything by itself. It needs to cooperate with the other agents and to count on their help to get better results. Many works concerning the concepts of negotiation and cooperation are realized and the research in this field remains very active [42]. The economic theories constituted a good source of inspiration (Contract Net Protocol, auctions, etc.) [13]. Let us note the interdisciplinary aspect of the MAS which is, in our opinion, a big strength allowing to build algorithms based on well-founded and widely tested theories from other fields. To guarantee end to end QoS, the cooperation between routers is essential. It is thus possible to rely on the works developed in the multi-agent domain to allow the routers to reach agreements;

- ***Adaptability:*** In order to provide more flexibility, researchers are interested in the learning capability in multi-agent systems. A part of the researches is focused on genetic algorithms [5], while the others use the reinforcement learning [16], etc. By endowing the agents with capabilities of learning, it will be possible to face unexpected situations. If we return to the autonomic networks, using agents having capacities of learning can be very beneficial and allows adapting more effectively to the evolutions of the networking domain.

2.2 Related Work

The use of agents and multi-agent systems in the telecommunication area is also a great challenge. The complexity of the networking domain has been an obstacle for the direct employment of artificial intelligence techniques.

2.2.1 Agents and telecommunication area

Several researchers coming from the two communities, networks and artificial intelligence, were interested by introducing agents into telecommunication networks, but they had different motivations. The AI community sees in networks a good domain of application of its techniques via multi-agent systems.

The network community took an interest in the agent technology because multi-agent systems, thanks to the characteristics of the agents including autonomy, pro-activity, reactivity, etc. are well suited to the management of distributed systems. Telecommunication networks are a very good example of such distributed systems. The majority of these works use the agents to solve particular problems (optimization of the routing, bandwidth management, etc.). The agents are, in general, introduced into the network to assure a particular control of the entities to which they belong. The topology discovery of a dynamic network by a certain number of mobile agents, sharing their information and cooperating in order to build optimal routing tables, had been treated by [33] and [39].

Another application of mobile agents consisted in using them for a better decentralized management of the network, which replaces the classic client-server model [40]. In a centralized management, the interaction between clients and server and the periodic interrogation of the MIB (Management Information Bases) generate an intense traffic which overloads the management station, generating problems of scalability. Mobile agents are, thus, a means to distribute and to scale the network management. The mobile agents decentralize the treatment and the control, and reduce consequently the traffic around the management station, distribute the treatment and increase the flexibility of the nodes management. In mobile networks, mobile agents can be used to register the profiles of the different users, and be deployed every time the subscriber asks for a given service [28]. The concept of VHE (Virtual Home Environment) is one of the aspects of the mobile networks in which the mobile agents were chosen as a solution [17, 28]. The mobile agents were also used for the detection of intrusions [21].

Ant agents were used for the optimization of the routing. Sigel *et al.* [44] used it for the management and the update of routing tables in a low orbit constellation of satellites. Ant agents have also been used for the fault detection [49]. Besides, Sugauchi *et al.* [46] used mobile gents, but without ant behavior this time, for the fault detection in end to end paths. Once the fault discovered, the agents inform the administrator, and he can, at this moment, focus only on paths containing problems.

Some contributions are, however, based on static agents to avoid the traffic overload which may appear with the deployment of a very big number of mobile agents in the network. In this regard, we can name the work of [7], who used agents based on the INTERRAP hybrid approach [34]. In this work, the role of the agents consists in maximizing the use of channels and in minimizing the probability of calls blocking in a mobile network. Dutta *et al.* [16] use Q-learning agents to estimate distributed bandwidth for effective routing and failure detection. A multi-agent system is used to manage LSP (Label Switched Path) in an MPLS (Multi Protocol Label Switching) [37]. An architecture based on a set of mobile and static agents was proposed in [8] to provide an inter-domain routing with quality of service. The deployed agents allow obtaining intra-domain paths coherent with the page 8 (15) inter-domain constraints. The adaptation of inter and intra-domain routing mechanisms is possible and may need negotiation between agents.

Jennings [29] also applied the economic theories to cooperative (static) agents in order to maximize the use of the network resources (bandwidth, loss of packets, etc.) by processes of negotiation [19]. The same theories were also used to assure the congestion control thanks to a group of mobile and static agents [50]. In [2], dynamic groups of agents are used to realize an adaptive monitoring of the network.

A distributed management based on a multi-agent architecture is proposed in [9]. Every agent possesses a set of skills allowing it to realize a given functionality in the management domain it belongs to. A skill can have as function the fault detection, the monitoring, etc. A central entity called Master Agent is responsible for the centralization of the activities of agents belonging to the same domain and to delegate to them the appropriate tasks.

2.2.2 Multi-agent platforms as a tool to provide the autonomy

Given the interdisciplinary aspect of the multi-agent systems, it is not surprising to find out different agent-based tools. Indeed, some tools were developed to demonstrate a particular aspect of a given problem. This kind of tools is used, in general, in an particular way by research teams and we cannot use them in a general context.

A large part of the available tools is fortunately aimed at a wider spectrum of use. These tools can be classified into two important categories:

- Simulation-oriented: in that case, the tool is seen as a virtual laboratory and allows testing the solution in a simulated mode before performing a real deployment. In general, we use multi-agent simulation when the classic simulators based on mathematical models are no longer effective. There is quite a few agent-based tools offering this possibility: Swarm[52] which is a software package for multi-agent simulation of complex systems (ecology, organism, etc.) [52], oRis [54] which is virtual reality oriented, Madkit [55] which is based on an organizational approach, etc.

- Conception oriented: the tool allows to create agents and to deploy them on a machine or a network of several machines. According to the available tool, it is possible to have an (graphic) assistance throughout the process of agent development. Among the most complete tools, we can name Zeus [56] and AgentBuilder [52], etc. Zeus is a toolkit for constructing collaborative multi-agent applications. It defines a multi-agent system design approach and supports it with a visual environment for capturing user specification of agents that are used to generate Java source code of the agents. Typical application areas of Zeus agents were expected to be task-oriented domains such as service provisioning, resource/process management, and supply chain management [36]. AgentBuilder is another toolkit that reduces development time and cost and simplifies the development of high-performance agent-based systems. It is currently available in two different versions to meet a wide variety of developer needs: Lite, an entry-level product for agent software developers, and Pro, for multi-agent development. AgentBuilder has been used in applications like: buying and selling Electric Power in a Dynamic Auction, private email, shopping agents, etc.

There are also platforms developed to assure the agents' mobility, some examples can be: Grasshopper [3], Aglets [57], Voyager [58], etc. These tools are interesting only for applications requiring the mobility of agents and have to manage the security (of agents and of hosts) feature.

A good comparison of the most known available agent-based tools can be found in [38, 41].

2.2.3 Network simulation platforms

As it has been extensively described before, many multi-agent researchers proposed their own multi-agent platform. These platforms naturally fit to multi-agent simulations issues. When one wants to adapt multi-agent systems to networking issues, one has to choose between using either existing simulation contexts, or developing new ones. Considering network issues, the mechanisms are very complex, and high level issues rely on many technical data. Thus, in a network context a network simulation platform is preferred and to add some agent components. The platform which seems adapted to our context is called JSIM. This network simulation platform is made out of components, which are completely modular. Integration of new protocols, mechanisms, network elements, and even agents, is made possible by this modular architecture. J-Sim is an open component-based simulation environment. Structure of network nodes is described in a generic way, using a set of components, including a CSL component, which provides basic network services at physical, link, and network layers (Figure 2.1). Over that CSL are plugged a set of components from rotocol, transport, and application layers. These components provide several transport, routing, and signaling features. Components interactions are defined in terms of contracts between them. Requests and responses are passed via components "ports", and are defined by service contracts. This platform integrates all the "network" components and we need to add the "agent" component useful for our purpose.

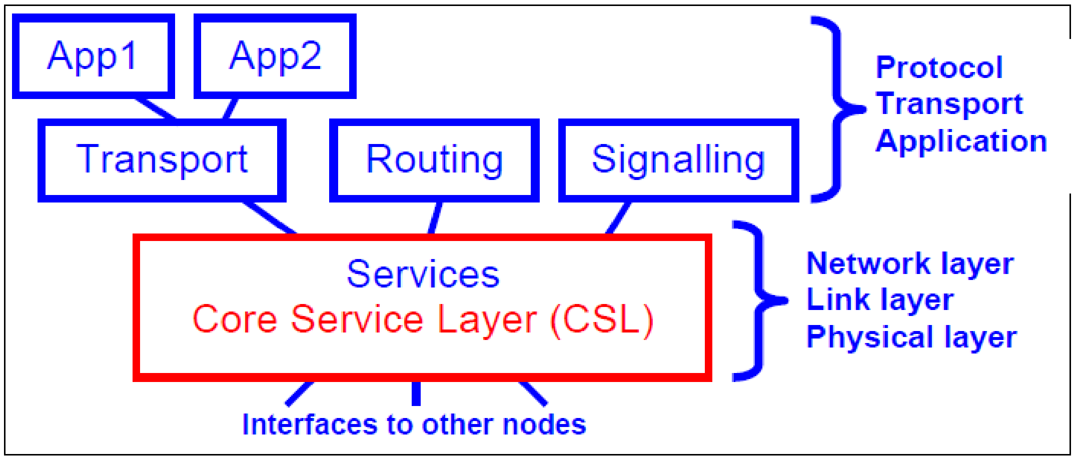

Figure 2.1 - A generic node description in the J-Sim environment

2.2.4 Integration of an agent module in a JSIM environment

In this section we explain the architecture and mechanisms developed for an agent application in a JSIM context.

Agent framework
An agent is made of a decision core and several modules (Figure 2.2). Decision core is the kernel of the application. It is based on synchronous execution of several "behavior units", and is able to compute received and observed data, in order to decide on actions to execute.

Modules are the sensors and effectors of the agent. These are interfaces with the environment of the agent, which allows the behavior units to observe a given part of the functional environment (reading an element in a MIB, receiving inter-agent messages, etc…), and act on it (configuring equipments, send inter-agent messages, reporting events to the network operator, etc.).

In this section the decision core and the modules mechanism are described in details.

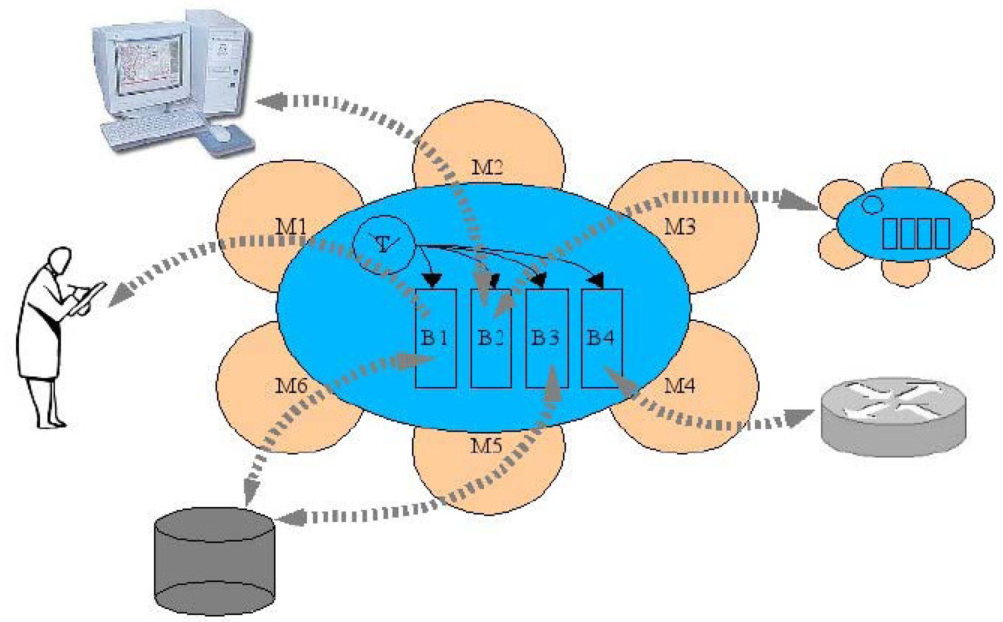

Figure 2.2 - The agent within its environment

Decision core
The decision core is based on one or more behavior units. These units are execution units, synchronized by an internal timer. They are executed in a given sequence, at a fixed rate.

There are three different kinds of behavior units:

- Observation and listening units, which are able to gather information about network elements and neighbor agents;
- Computing units, which extract useful information from the data gathered by observation units, and compute a “good” decision for the network element configuration;
- Action units, which apply actions decided by the computing units.

These units are executed sequentially, in order to plan mechanisms over several behavior units. Decisions are taken after information was collected. Each one of these behavior units can access the modules they want. Actions decided by these units can use primitives of available modules (“send a message”, “check a mailbox”, “set a parameter”, etc…). Each of these can access the controlled equipment, send messages to the neighbors, and update a remote database.

Modules

Modules are interfaces between an agent and its environment. Primitives are described and behavior units can access them. Development of these modules is strictly separated from the development of the application. It depends on the environment (architecture, hardware, technical means, etc…) and on primitives they provide.

Agent interfaces

These interfaces are classified into 3 categories (Figure 2.3): Agent-to-router interfaces, agent-to-agent interfaces, agent-to-human interfaces.

Agent -to -router interface

Agent-to-router interface is the sum of the features provided by the network environment to the agent behaviors. In a simulation context, this interface relies on the facilities the simulation environment provides for the integration of new components. If just a small number of interactions between the agent and the network features is available, the efficiency of agents will be reduced. On the contrary, if agents can monitor many variables and tune a lot of parameters, their efficiency will be increased.

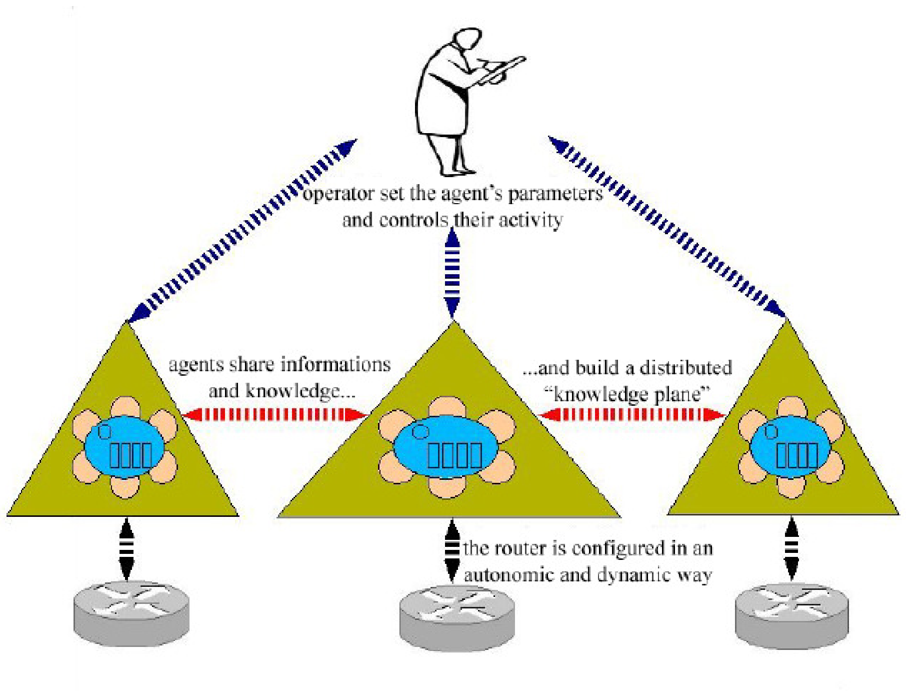
operator set the agent's parameters and controls their activity
agents share informations and knowledge...
...and build a distributed "knowledge plane"
the router is configured in an autonomic and dynamic way

Figure 2.3 - Interfaces between the agent and its environment

Agent -to -agent interface

Agent-to-agent communication is a key issue, and one of the most important innovations provided by multi-agent systems. Transversal sharing of information by agents allows to build a global distributed “knowledge plane” we describe in the second section. Each agent can build its own situated view of the network state. Addition of these different situated views results in a distributed knowledge plane. The situatedness of agent knowledge is a key point in our architecture: it ensures that our system can be scalable.

This transversal, peer-to-peer, and not hierarchical communication model is an answer to one of the most important issues in the networking domain: how to build and maintain a global network state representation, without running into scalability problems? Building in real time a unique and centralized view of the network is only possible in very small networks. On the contrary, if we accept that this representation is not centralized, not always complete and deterministic, our architecture is a suitable answer.

In order to exchange data in a coherent way, a suitable modeling of these data has to be adopted. On the suitability of this model will depend the efficiency of the application. It is necessary, for example, to find an adequate level of granularity.

Exchange protocols can also be a key issue of an application. Depending on the kind of protocol used, few messages can exchange a huge amount of information. These protocols also have to be extremely robust: each agent has to be autonomic, it is a key quality of our system. The stability of an agent can never rely on the reception of a message. Every situation must be faced and managed. One should avoid explicit requests. Instead, one way information messages are preferred.

Agent -to -human interface

The agent-to-human interface is a monitoring interface. This interface gathers all agent-monitoring and agent-configuring facilities. Within a simulation environment, it also relies on the monitoring facilities of the environment: configuring can either be done off-line, before the simulation, or on-line, during the simulation, or both; monitoring can be done on-line during the simulation, or off-line, after the simulation (visualization of the results), or both on-line and offline. We will naturally prefer an on-line configuring, on-line monitoring simulation environment, which allows on-line configuration and on-line monitoring of agents.

Chapter 3

Agent applications

3.1 OSPF-Weight-Setting

We developed several agent applications within this agent framework. One of them is an OSPF-Weight-Setting application and is the purpose of this paper.

OSPF is a dynamic routing protocol that uses weights affected to each interface, to route the traffic in a more balanced way. The purpose is to find and set a nearly optimal configuration of OSPF weights within an OSPF area. Many research teams worked on this issue, and a huge amount of OSPF-weight-setting heuristics has been proposed. These heuristics are generally either off-line,
or centralized. We think that a distributed approach, with no need for global information, would be more scalable, and thus more realistic. Our purpose is not to find new heuristics, but rather to provide a new distributed and situated approach, and to adapt common heuristics to this approach, in order to make them more realistic.

Here is a summary of an agent behavior example for this application:

- ***Agent monitors the different queues of the router on which it is embedded.*** An agent polls at a fixed rate the number of packets in each queue. This is achieved using a special J-Sim control module from the agent testbed.
- ***Agent shares precomputed knowledge about the filling of its queues with its neighbors.*** An agent extracts useful data from the raw information it monitored from network features, and broadcasts this data to its nearest neighbors ("nearest" means "within a distance from 1 to 3 hops").
- ***Agent decides of the weight it should set for each interface.*** Each agent computes the knowledge it gathered to decide for each interface if the weight should be left unchanged, increased, or decreased. This decision is influenced by several factors including the current state of local interfaces, compared with usual states of these interfaces; the current state of the neighborhood, compared with its usual state, etc.
- ***In case of a change of weights, agent broadcasts its decision to its neighbors.*** An agent deciding to change the weights of its interfaces should warn the neighbors of this decision. Thus, the neighbors do not take the same decision at the same time. This mechanism allows an easier and faster convergence of the system.

This application with such behaviors was developed, tested, and proven to be efficient. It also provided good results in terms of traffic engineering performance and system convergence.

Agents behavior
The implemented behavior can decide of 2 independent actions: on the one hand, it can decide to change the weight of one of its interfaces; on the other hand, it has to decide on the new weight to set. These decisions have to be separated, because changing a weight causes the whole network to be flooded with the new weights topology, and all the routing tables to be recomputed. It is then important to decide very seldom to change the weights. This decision can rely of different pieces of information as the new weight to set.
The choice of interface weights is based on the current load of these interfaces, and on the state of the routers directly linked to these. It is indeed very important, before deciding to redirect more traffic on an interface, to see if the adjacent router to this interface is already loaded or not. In the following example (Figure 3.1), router-agent O has to choose OSPF weights of its interfaces.

Available information is showed on the figure: the load of each one of its interfaces (cursors near each interface), and the state of nodes directly linked to these interfaces (color of the nodes). In this case, router-agent O has to set the weight of 3 interfaces, numbered from 1 to 3. For the interface 1, the current load of the interface is high, and the adjacent router (A) is in a "critical" state (plain). Agent O's decision will then surely tend to increase this interface weight, in order to reduce the traffic passing through it.
For interface 2, the load is high, but the state of the adjacent router (B) is "good" (void), so we can guess that the weight of this interface can remain the same. For interface 3, the load is low and the state of the adjacent router is "good", then the decision may be to decrease the weight associated to this interface.

On the contrary, deciding to effectively change the weights of the interfaces in a router (and, then, to reset the whole OSPF protocol in the network) does not rely directly on the load of interfaces, but on the global state of the router. This state takes into account the load of interfaces (max load, average load, etc.) and an evaluation of the state of the environment (state of the neighbor routers). To avoid multiple simultaneous OSPF resets in the same neighborhood, each agent sends an "I reset!" message to all its neighbors (on 1, 2, or 3 hops). Multiple resets in a same neighborhood could indeed cause oscillation and strongly decrease the network performance. To avoid these problems, we also propose to take into account 2 kinds of influences: stabilizing
influences and destabilizing influences. A positive state of the neighbors is a stabilizing influence while a negative state of the neighbors is a destabilizing influence. Heavy load of an interface is a destabilizing influence, when light load of an interface is a stabilizing influence.

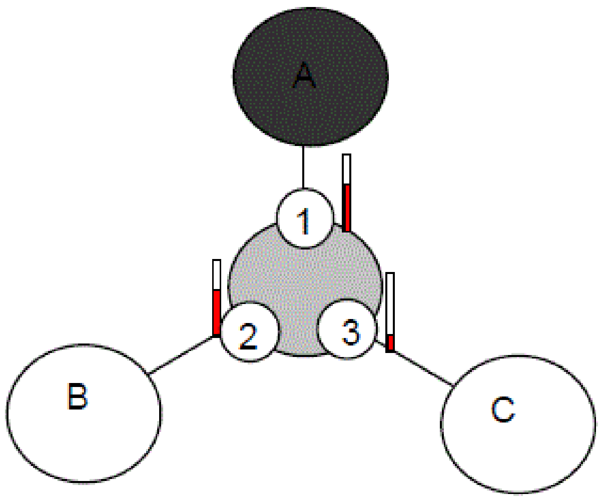

Figure 3.1 - An example of a weight setting situation

To avoid oscillation around a not-perfect-but-optimal configuration, we included within the agent behavior a habituation phenomenon. That means that a data has no sense by its own. Each data taken into account by the agent is pre-computed to compare it with previous identical data, to know if it is "particularly high", "normally high", "surprisingly high", etc… More precisely, each data is compared with its minimum, its maximum, and its average. This pre-computing is done by normalization (Figure 3.2). Our normalization is done with the following system:

Xnorm = [(X – Xmin)/Xavg]*0.5 if $0 \leq X < Xavg$
Xnorm = 0.5 + [(X – Xavg)/Xmax]*0.5 if $Xavg \leq X < Xmax$

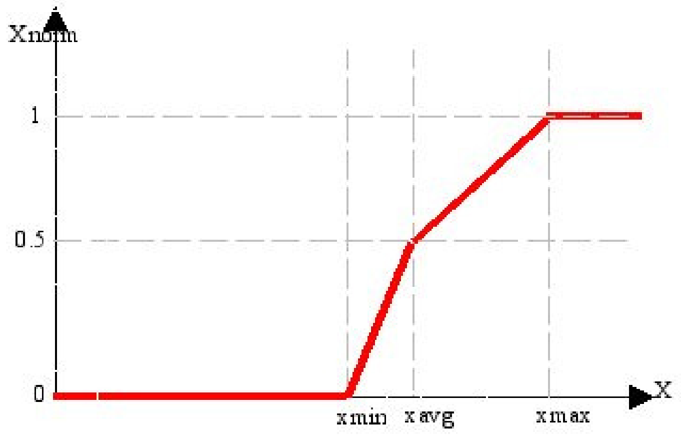

Figure 3.2 - A representation of our normalizing operation

3.2 Agent application in PUC-RJ

Some of the mechanisms used for the implementation of the agent's properties described previously are:

- **Self-organization**

  Self-organization is defined in [43] as the mechanism or processes that allow a system to cope with the dynamism, growth, distribution, complexity and dynamic changes in requirements, at runtime, without an explicit external control.

- **Norms**

  Multi-agent systems are open societies in which autonomous entities, heterogeneous and independently designed can work for similar or different purposes. In order to deal with the heterogeneity, autonomy and diversity of interests among the different members, such systems provide a set of standards that are used as a control mechanism to ensure a desirable order in which agents can work together [18].

  These rules govern the behavior of agents by defining obligations (indicating that the agents are forced to do something), permissions (indicating that agents are authorized to act in a particular way) and prohibitions (indicating that agents are prohibited from acting a certain way). Moreover, rules can provide stimulus for its accomplishment using the definition of rewards and may discourage its violation, establishing punishments.

- **Thrust**

  Trust is security; it is the assurance that an individual has faith in the probity (honesty, trustworthiness, honesty) of someone [25].

- **Reputation**

  Reputation can be understood as the social evaluation (opinion) that an entity has about another entity or group of entities or organization [25].

Work Developed in LES

As a result of dissertations, doctoral and post-doctoral theses by the LES group, important contributions have been made under the various concepts, such as norms, reputation, trust, self-healing, self-configuration and self-organization. Brief descriptions of some of these works are described below:

- **Norms**: A language to represent norms in which is possible to specify the entity responsible for fulfilling a determined norm, and the rewards and punishments for fulfilling or not the norm, respectively. Such language can be seen in [47].

- **Reputation and Trust**: DRPMAS [35], a recommendation *framework* that provides support to implement different methods and models of reputation and trust.

- **Self-***(Self-healing, Self-configuration, Self-configuration, Self-optimization and Self-organization)
  - **a.** JAAF [35], a framework that enables the implementation of mechanisms able to collect information about the execution of a specific resource, analyze such

information in order to detect problems and discover solutions, select the better solution and, finally, perform the solution selected.

b. An engineering method based on the simulation to support the project, development, simulation, validation and refactoring of self-organizing multi-agent systems and architecture based on simulation. Such work can be seen in [23].

c. The GenArch [10] is a software product line derivation tool based on a model-driven approach that supports the development of self-adaptive systems by expressing adaptations and its variability in high-level languages and the automatic derivation of adaptation actions from architectural models.

3.2.1 Details of the JAAF Framework

The JAAF framework is implemented by the use of software agents, as illustrated in Figure 3.3 by the JAAF architecture. JAAF extends JADE a FIPA compliant framework to implement multi-agent systems (MAS) developed in Java, in order to represent three concepts: (i) agents that perform self-adaptation, (ii) plans executed by agents representing self-adaptation processes (or control loops) and (iii) activities that are the steps of such processes. In order to implement self-adaptive systems it is necessary to instantiate the JAAF framework by implementing the plans or control loops and their activities. JAAF already provides a control loop composed of four activities.

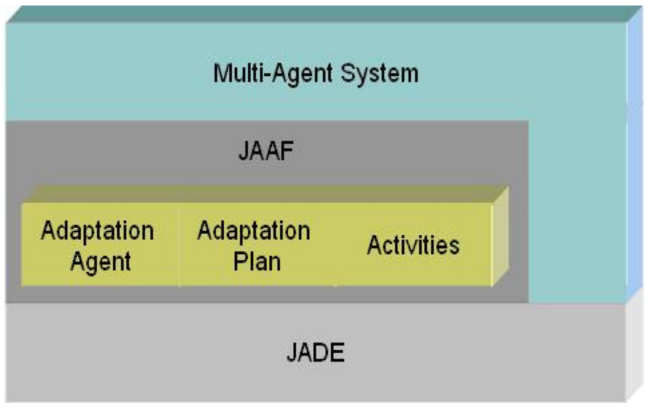

Figura 3.3 - JAAF Architecture

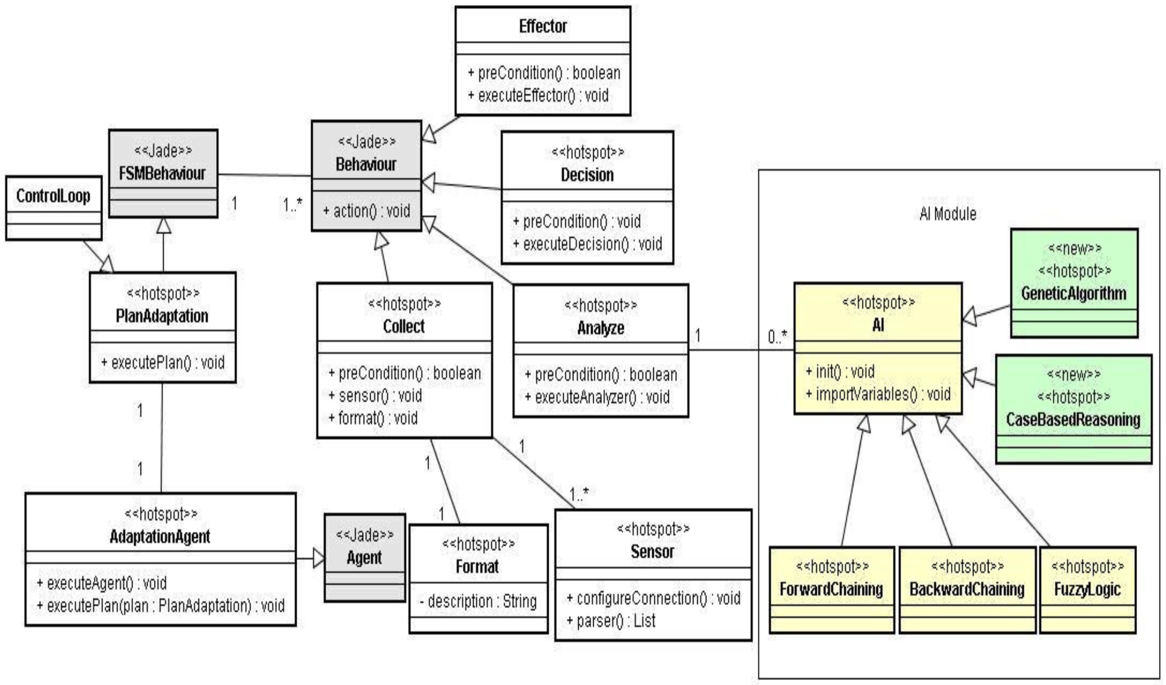

Figura 3.4 – Class Diagram of JAAF

The class diagram depicted in Figure 3.4 illustrates the main JAAF classes. The self-adaptive agents are represented by the *AdaptationAgent* class and the self-adaptation process by the *PlanAdaptation* class. Such class extend the JADE class *FSMBehaviour* that provides support to the implementation of finite automata composed of activities (or behaviors) represented by the *Behaviour* class.
JAAF already provides a self-adaptation process represented by the *ControlLoop* class that is composed of four activities (Figure 3.5): Collect, Analyze, Decision and Effector, each one extending the *Behaviour* class. This control loop was created based on the self-adaptation process [15]. Note that, although JAAF already provides such control, it is possible to define others from the *PlanAdaptation* class. It is also possible to implement different activities (or steps of the loop), from the *Behaviour* class, to different control loops.

COLLECT

This is the first step executed by the process. It is responsible for providing mechanisms to collect, aggregate and filter (format) data collected from the application.
In order to represent this idea, the framework offers two concepts: sensor and format. The *sensor* defines the place where the data should be collected (database, log, etc) and the *format* defines the format of the collected data. This activity also specifies *when* the agents should collect the data, i.e., it describes the preconditions to activate the sensor.

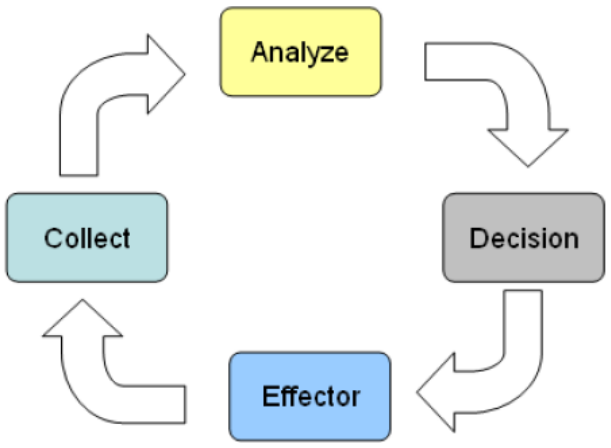

Figura 3.5 - Control loop provided by the JAAF framework

ANALYZE

The analyze activity is responsible for providing mechanisms that analyze the data collected in the previous activity in order to detect problems and suggest new solutions. The framework gives support to three techniques: (i) rule based reasoning (forward chaining, backward chaining and fuzzy logic), case based reasoning and genetic algorithm.

DECISION

Decision is the third activity responsible for deciding which action (or behavior) will be the next one to be executed by the agent, while trying to achieve the goal. Such decision is based on the information provided by the previous step.

EFFECTOR

This is the last step of the self-adaptation process. It receives the selected action from the *Decision* activity, and informs the agent the action to be executed. When the action is executed, the control loop can be executed again whether any self-adaptation is necessary.

HOT-SPOTS AND FROZEN-SPOTS

Since JAAF extends JADE, the JADE kernel is also the kernel of JAAF and the hot-spots of the JADE are the hot-spots of JAAF. For instance, the process used by agents to communicate, and the agents' identifiers are examples of JAAF hot-spots inherited from JADE.
The hot-spots specifically defined in JAAF are:

1. Agent (*AdaptationAgent* class): By extending such class and implementing the *executedPlan* method, it is possible to define different algorithms to execute the plans of an agent.
2. Plan of self-adaptation (*PlanAdaptation* class): It is possible to define new control loops (or plans) and the sequence to execute the activities of the control loops. JAAF already provides a default control loop implemented in the *ControlLoop* class.
3. Activities (*Behaviour* class): It is possible to define new activities to be called by the control loops by extending the *Behaviour* class. JAAF already offers four activities (Collect, Analyze, Decision and Effector).
4. Sensor (Sensor class): One can define *when* and *where* the data should be collected.
5. Format (Format class): It is possible to define the format of the data to be collected by the Sensor.
6. Intelligent Algorithm module: JAAF offers three kinds of algorithms: rule-based reasoning (forward chaining, backward chaining and fuzzy logic), case-based reasoning

and genetic algorithm. Such algorithms can be used at any point of the system to help with the self-adaptation.

As mentioned above, in order to implement a self-adaptive agent, the following steps should be performed: (1) define a plan of self-adaptation by extending the *PlanAdaptation* class; (2) create the activities that compose such plan by extending one of the JADE classes that represent behaviour; and finally (3) create a self-adaptive agent by extending the *AdaptationAgent* class.

3.2.2 JASOF

Due to the complexity of building self-organizing systems, [14, 22] have elaborated a series of recurrent design patterns that are commonly present in these systems. The aforementioned patterns aim to smooth the process of constructing self-organizing systems by offering a set of directions and making adequate reuse of solutions that have already been tested and documented. In this context, the JASOF framework, which is based on the BDI paradigm extending Jadex, provides the necessary infrastructure to implement the patterns and the environment where the agents operate. The patterns in JASOF are provided as a set of goals and plans encapsulated as Jadex capabilities, which can be easily imported and implemented by an agent.

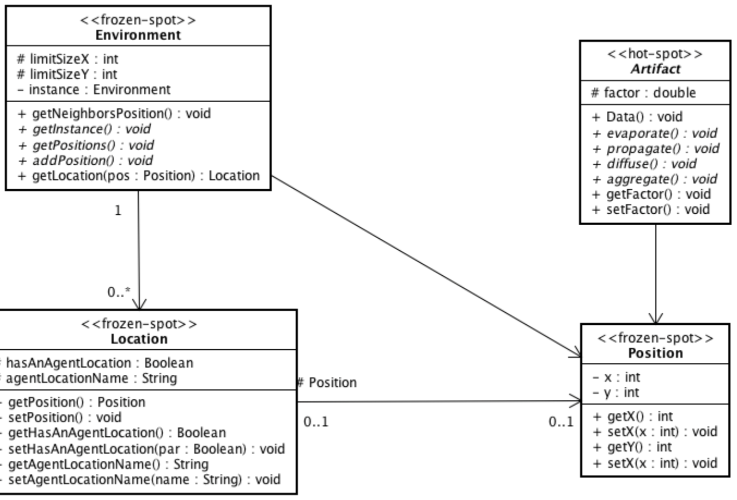

Figura 3.6 - JASOF Class Diagram - Environment

The environment is modeled as a space composed of interconnected positions following the notion of a grid. Furthermore, it is based on the Agents & Artifacts approach, which considers the environment populated by agents and artifacts, where artifacts are reactive agents providing resources and services, meaning an active environment. The class diagram in Figure 3.6 shows the structure of the environment, where the class *Environment* contains information about the available locations, the agents in the system and your neighborhood. These locations are represented by the class *Location* that provides information about its artifacts (class *Artifact*), their manager agent and their relative position in the environment (class *Position*).
Each location has an agent, called agent *Location*, which is responsible for managing the artifacts stored in this location, as well as for executing the patterns and providing access to artifacts for agents in its location.

JASOF's Structure

First, we show how the environment acts in our framework through the agent Location. Then, the plans provided by JASOF that enable the implementation of the referred patterns are described in details. Figure 3.7 shows the class diagram that represents these plans and the hot-spots from which it is possible to create new patterns not yet documented in the literature.

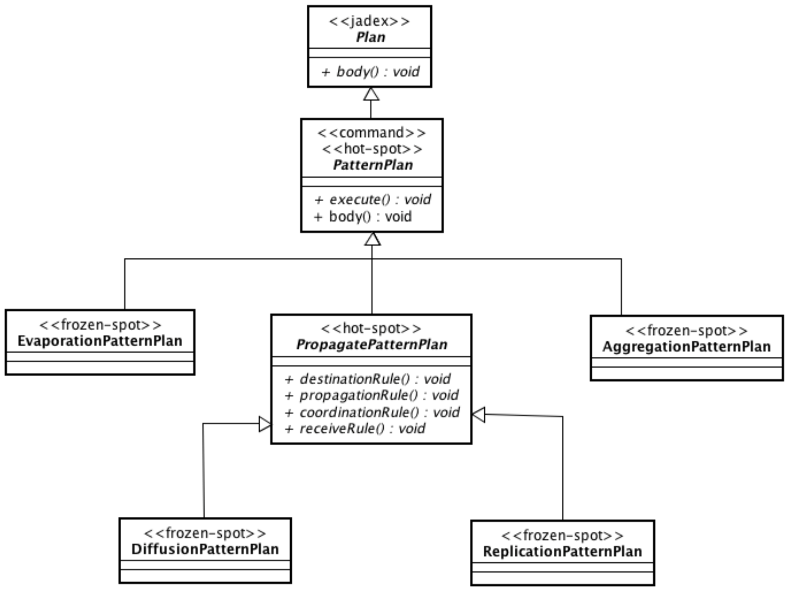

Figura 3.7 - JASOF Class Diagram – Plans

Agent Location

Each location in the environment has an agent, called agent *Location*, responsible for managing the artifacts stored in this location, to execute the patterns and to provide artifact access to agents that are in the same position. It is through the agent Location that the environment has an active behavior, executing and interacting with the other participants of the system. Those participants are comprised of all agents in the environment, Locations and others. The agents that are not *Location* are called *User* agents in the framework.

Through communication with the agent *Location* of a specific location it is possible for a given agent to insert an artifact, which can represent relevant information such as a pheromone, or to read the stored artifact. All communication with the agent *Location* to insert and read is a frozen-spot of the framework. The reason for associating agents to each location is justified by the need to search for forms of decentralization, despite the large number of agents and the communication required for reading and writing information. In the following sections the ease the environment and agents have in implementing self-organization patterns can be observed.

Diffusion Pattern

The idea behind the Diffusion Pattern is the process present in nature that when a pheromone is deposited into the environment it spontaneously tends to diffuse to neighboring locations [48]. Diffusion distributes the information equally across all neighboring locations.

In order to implement the diffusion pattern it is necessary for the agent to import the *DiffusionPattern* capability. It provides the infrastructure needed for an agent to send information, decreased by a factor, to a set of neighbors selected according to a rule (*PropagationRule*) and perform the actions related to the coordination process or react to a received message. This is, in essence, the coordination mechanism. Such capability has the class *DiffusionPatternPlan*, which needs to be extended in order to define the rules for selection of agents to receive the message (method *DestinationRule*), the actions of a coordination (method *CoordinationRule*) and the propagation (method *PropagationRule*) process. The message *diffusionMsg* is the standard event used to perform the diffusion pattern, so importing that capability implies that the agent can receive and handle that message. If necessary it is possible to modify the *ReceiveRule* method and change this process.

Evaporation Pattern

Evaporation can be considered a mechanism to reduce the information amount, based on a time relevance criterion, avoiding an overload of information.
The Evaporation pattern can be implemented by an agent, using JASOF, by importing the *EvaporationPattern* capability. Such capability will periodically execute the *evaporate()* method, present in the class *Artifact*, of the instances contained in the Belief Base of the respective agent *Location*. To change the frequency it is necessary to modify the parameter *recurdelay* that by default is 6000. Another parameter that can be determined is how much of the information will be evaporated per cycle; in this case the attribute *factor* of the *Artifact* class must be changed.

Aggregation Pattern

Aggregation is a mechanism of reinforcement and is also observable in human social tasks [51]. When redundant information is stored their relevance increases.

As in the other patterns, to implement the Aggregation patterns it is necessary to import the *EvaporationPattern* capability, which capture the concept of Evaporation, periodically check the artifacts saved in the Belief Base and check which of them may suffer an aggregation. This aggregation is accomplished by the method *aggregate()* of the *Artifact* class. An aggregation process between two artifacts produces one with more relevance.

Replication Pattern

Natural systems typically feature replication mechanisms in order to increase security and robustness. The Replication pattern works in a way similar to the Diffusion pattern; the main difference resides in the factor of the information replicated. In the diffusion case, the artifact suffers a decrease in the factor attribute and, on the other hand, the replication does not modify that attribute, keeping its actual value. The mechanisms *DestinatioRule*, *CoordinationRule* and *PropagationRule* are used for the same purpose.

3.2.3 JAAF and JaSOF with Ginkgo in an Example

The frameworks JAAF and JASOF were developed on the Jade and Jadex platform, respectively. Nowadays, we are in the state of migrating such frameworks to the Ginkgo platform, where the main advantage for using this platform is in the fact that Gingko provides support for work with network devices and exhibits a satisfactory performance when compared with Jade and Jadex platform.

In order to validate the frameworks JAAF and JASOF we are developing an autonomic management system based on information of the objects managed (also known as MIB-II) in the network.

In the self-adaptation level, the agents implemented by JAAF are based on a self-adaptation control-loop that is used to collect information from the network devices, analyze such information aiming to detect errors, in this case the rule-based reasoning is used to detect errors and the case-based reasoning to suggest solutions, and, next, decide what solutions (adaptation strategies) must be applied in order to solve such errors.

We are using the JASOF in order to develop mechanisms that use self-organization patterns, for example the Replication, Diffusion, Evaporation and Aggregation pattern, to enable the different network devices share decentralized information. Considering that some such patterns are provided by Ginkgo, little effort was necessary to include the self-organization idea. For example, some changes in the importance of the data in the network. In the application mentioned above, we are performing tests in order to validate the decentralized file configurations and routing between the devices, such as routers.

Bibliography

[1] Aknine, S., Pinson, S., Shakun, M.F. “An Extended Multi-agent Negotiation Protocol”. *International Journal on Autonomous Agents and Multi -agent Systems*, Sycara, K., Wooldridge, M. (eds.), Vol. 8 no.1, pp.5-45, 2004.

[2] Al Shaer, E. “A Dynamic Group Management for Scalable Distributed Event Correlation”. *IEEE/IFIP Integrated Management (IM'2001)*, May 2001.

[3] Bantz, D. F., Bisdikian, C., Challener, D., Karidis, J. P., Mastrianni, S., Mohindra, A, Shea, D. G., Vanover, M. Autonomic personal computing, *IBM Sys* J 42(1) pp. 165-176, 2003.

[4] Baümer, C. and Magedanz, T. “Grasshopper – A Mobile Agent Platform for Active Telecommunication**”.** *Third International Workshop on Intelligent Agents for Telecommunication Applications. LNCS 1699*, pp 19-32, 1999.

[5] Berger, M. and Rosenschein, J.S. “When to Apply the Fifth Commandment: The Effects of Parenting on Genetic and Learning Agents”. *AAMAS’2004*, New York, USA. ACM, pp 19-23, July 2004.

[6] Brooks, R. A. “A Robust Layered Control System for a Mobile Robot”. *IEEE Journal of Robotics and Automation*, Vol. 2, No. 1, March 1986, pp. 14–23; September 1985.

[7] Bodanese, E.L. and Cuthbert, L.G. “A Multi-Agent Channel Allocation Scheme for Cellular Mobile Networks”. *ICMAS’2000, Boston, USA. IEEE Computer Society press*, pp 63-70, July 2000.

[8] Calisti, M. and Faltings, B. “A Multi-Agent Paradigm for the Inter-Domain Demand Allocation Process”. *DSOM'99, 10th IFIP/IEEE International Workshop on Distributed Systems, Zurich, Switzerland. LNCS 1700*, pp 76-88, October 1999.

[9] Cheikhrouhou, M., Conti, P., Marcus, K. and Labetoulle, J. “A Software Agent Architecture for Network Management: Case-Studies and Experience Gained”. *Journal on Network and Systems Management*. Vol. 8(3), 2000.

[10] Cirilo, E.J.R., Nunes, I., Kulesza, U., Lucena, C.J.P. (2009), Automating the Product Derivation Process of Multi-Agent Systems Product Lines, XXIII Simpósio Brasileiro de Engenharia de Software (SBES 2009), Fortaleza, Brasil, pp. 12-21.

[11] Clark, D., Partridge, C., Ramming, J.C., Wroclawski, J.T. “A Knowledge Planefor the Internet”, *Proc. Applications, technologies, architectures, and protocols for computer communication, ACM SIGCOMM 2003*, Karlsruhe, 2003.

[12] Costa, A.; Lucena, C. J. P.; Silva, V. T.; Cowan, D. ; Alencar, P. A hybrid diagnostic-recommendation system for agent execu-tion in multi-agent systems. In: ICSOFT 2008 – 3rd International Conference on Software and Data Technologies, Porto, Portugal, 2008.

[13] David, E., Azulay-Schwartz, R. and Kraus, S. “Bidders` strategy for Multi-Attribute sequential English auction with a deadline”. *AAMAS -2003*, Melbourne, Australia. ACM, July 2003.

[14] De Wolf, T. and Holvoet, T. Decentralised Coordination Mechanisms as Design Patterns for Self-Organising Emergent Applications., Proceedings of the Fourth International Workshop on Engineering Self-Organising Applications, Editors: S. Brueckner, S. Hassas, M. Jelasity, D. Yamins, Future University-Hakodate, Japan, pp 40-61, 2006.

[15] S. Dobson, S. Denazis, A. Fernández, D. Gaiti, E. Gelenbe, F. Massacci, P. Nixon, F. Saffre, N. Schmidt, and F. Zambonelli. A survey of autonomic communications. *ACM Transactions Autonomous Adaptive Systems (TAAS)*, 1(2):223{259, December 2006.

[16] Dutta, P.S., Jennings, N. and Moreau, L. “Cooperative Information Sharing to Improve Distributed Learning in Multi-Agent Systems”. *Journal of Artificial Intelligence Research*, Vol. 24, pp 407-463, 2005.

[17] Esmahi, L., Impey, R. and Liacano, R. “Toward a Mobile Work Environment”. *2nd International Workshop*, *MATA'2000*, Paris, France. LNAI 1931, pp 30-47, September 2000.

[18]. López y López, F. Social Power and Norms: Impact on agent behavior. PhD thesis, University of Southampton.

[19] Faratin, P., Jennings, N.R., Buckle, P. and Sierra, C. “Automated Negotiation for Provisionning Virtual Private Networks Using FIPA-Compliant Agents”. *PAAM 2000*, Manchester, UK, pp 185-202, 2000.

[20] Ferber, J. “Multi-Agent System: An Introduction to Distributed Artificial Intelligence”. *Harlow: Addison Wesley Longman*, 1999.

[21] Foukia, N. “IDReAM: intrusion detection and response executed with agent mobility architecture and implementation”. *AAMAS’2005*, Utrecht, Netherlands. ACM, pp 264-270,cJuly 2005.

[22] Gardelli, L., Viroli, M. and Omicini, A. Design patterns for self-organizing multiagent systems. In Tom De Wolf, Fabrice Sare, and Richard Anthony, editors, 2nd International Workshop on Engineering Emergence in Decentralised Autonomic System (EEDAS) 2007, pages 62-71, ICAC 2007, Jacksonville, Florida, USA, June 2007. CMS Press, University of Greenwich,London, UK.

[23] Gatti, M. A. C. Engineering of Self-Organizing Emergent Multi-Agent Systems: A Design Method and Architecture. PhD Thesis. PUC-Rio, 2009.

[24] Ganek (A.), “Autonomic Computing: Implementing the Vision”, Keynote presentation at the Autonomic Computing Workshop, *AMS 2003*, Seattle, USA, June 2003.

[25] Guedes, J., Silva, V., Lucena, C.J.P.: A Reputation Model Based on Testimonies. In: Proceedings of Workshop on Agent-Oriented Information Systems at CAiSE, pp. 37-47 (2006).

[26] Gelenbe, E., Gellman, M., Lent, R., Liu, P., Su, P. "Autonomous smart routing for network QoS", *Proc. First International Conference Conference on Autonomic Computing*; pp 232-239, New York, May 17-18, 2004.

[28] Glitho, R.H., Lenou, B.E. and Pierre, S. "Handling Subscription in a Mobile Agent Based Service Environment for Internet Telephony: Swapping Agents". *Second International Workshop on Mobile Agents for Telecommunications Applications (MATA'02)*, Paris, LNCS 1931, pp 49-66, September 2000.

[28] Hagen, L., Mauersberger, J. and Weckerle, C. "Mobile Agent Based Service Subscription and Customization using the UMTS Virtual Home Environment". *Computer Networks Journal, Special Issue on " Agent technologies within networks and Mobile Communication Systems"*, Elsevier Publisher, Vol. 31, Issue 19, pp 2063-2078, 1999.

[29] Jennings, N.R., Faratin, P., Norman, T.J., O'Brien, P. and Odgers, B. "Autonomous Agents for Business Process Management". *International Journal of Applied Artificial Intelligence,* Vol. 14 Issue 2, pp 145-189, 2000.

[30] Jrad, Z. and Krief, F. "An Intelligent Interface for the Dynamic Negotiation of QoS in ARCADE". ACS*/IEEE International Conference on Pervasive Services ICPS'2004*, Beirut, Lebanon, July 2004.

[31] Kona, M.K. and Xu, C.Z. "A Framework for Network Management using Mobile Agents". *1st IEEE International Workshop on Internet Computing and E -Commerce (ICEC'01)*, San Francisco, California, USA, April 2001.

[32] Merghem, L., Gaïti, D. and Pujolle, G. "On Using Agents in End to End Adaptive Monitoring". *E2EMon Workshop, in conjunction with MMNS'2003*, Belfast, Northern Ireland, LNCS 2839, pp 422-435, September 2003.

[33] Minar, N., Kramer, K.H. and Maes, P. "Cooperating Mobile Agents for Dynamic Network Routing". in *Software Agents for Future Communication Systems*, Chapter 12, Springer Verlag, pp 287-304, 1999.

[34] Müller, J.P. and Pischel, M. "Modelling Reactive Behaviour in Vertically Layered Agent Architecture". *ECAI'94*, Amsterdam, Netherlands. John Wiley & Sons, pp 709-713, 1994.

[35] Neto, B.; Costa, A.; Neto, M.; Silva, V.; Lucena, C. Jaaf: A framework to implement self-adaptive agents. In: Proceedings of the 21st International Conference on Software Engineering and Knowledge Engineering (SEKE'2009), 2009.

[36] Nwana, H. S., Ndumu, D. T. and Lee, L. C. "ZEUS: An Advanced Tool Kit for Engineering Distributed Multi Agent Systems", *PAAM'98*, ACM Press, pp 377-391, March 1998.

[37] Rahim-Amoud, R., Merghem-Boulahia, L. and Gaiti, D.
"Improvement of MPLS Performance by Implementation of a Multi-Agent System". *Intelligence in Communication Systems – IntellComm*, Montreal, Canada. Springer-IFIP, pp. 23-32, October 2005.

[38] Ricordel (P.M.), Demazeau (Y.) “From Analysis to Deployment: A Multi-agent Platform Survey”. *Engineering Societies in the Agents World, First International Workshop*. LNAI 1972, pp 93-105, 2000.

[39] Roychoudhuri (R.), Bandyopadhyay (S.) and Paul (K.) “Topology discovery in ad hoc Wireless Networks Using Mobile Agents”. *2nd International Workshop, MATA'2000*, Paris, France. LNAI 1931, pp 1-15, September 2000.

[40] Rubinstein (M.), Duarte (O.) and Pujolle (G.) “Evaluating the Network Performance Management based on Mobile Agents”. *LNCS 1931*, pp 95-102, 2000.

[41] Sabas (A.), Delisle (S.) and Badri (M.) “A Comparative Analysis of Multiagent System Development Methodologies: Towards a Unified Approach”. *Third International Symposium "From Agent Theory to Agent Implementation" (AT2AI-3), Sixteenth European Meeting on Cybernetics and Systems Research*, Volume 2, 599-604, 2002.

[42] Sandholm (T.) eMediator: “A Next Generation Electronic Commerce Server”. *Computational Intelligence* Vol. 18, no. 4, pp. 656-676, Special issue on Agent Technology for Electronic Commerce, 2002.

[43] Serugendo, G. Di M, Gleizes, M. P. and Karageorgos, A. "Self-Organisation in MAS", Knowledge Engineering Review 20(2):165-189, Cambridge University Press.

[44] Sigel, E., Denby, B. and Le Hégarat-Mascle, S. “Application of Ant Colony Optimization to Adaptive Routing in LEO Telecommunications Satellite Network”. *Annals of Telecommunications*, Vol. 57, N° 5-6, pp 520-539, May-June 2002.

[45] Sterritt (R.), Bustard (D. W.), “Towards an Autonomic Computing Environment”, *1st Int. Workshop Autonomic Computing Systems, DEXA'2003* Prague, pp. 694-698, 2003.

[46] Sugauchi (K.), Miyazaki (S.), Covaci (S.) and Zhang (T.) “Efficiency Evaluation of a Mobile Agent Based Network Management System”. *6st International Conference on Intelligence and Services in Networks, IS&N’99*, Barcelona, Spain. LNCS 1597, pp 527-535, April 1999.

[47] From the specification to the implementation of norms: an automatic approach to generate rules from norms to govern the behavior of agents. Autonomous Agents and Multi-Agent Systems, pages 113–155, 2008.

[48] Weyns, D., Holvoet, T., Schelfthout, K. and Wielemans, J. Decentralized control of automatic guided vehicles: applying multi-agent systems in practice. In *Companion To the 23rd ACM SIGPLAN Conference on Object-Oriented Programming Systems Languages and Applications* (Nashville, TN, USA, October 19 - 23, 2008). OOPSLA Companion '08. ACM, New York, NY, 663-674. DOI= http://doi.acm.org/10.1145/1449814.1449819.

[49] White (T.) and Pagurek (B.) “Distributed Fault Location in Networks using Learning Mobile Agents”. *2nd Pacific Rim International Workshop on Multi -Agents, PRIMA'99*, Kyoto, Japan. LNAI 1733, pp 182-196, December 1999.

[50] Yamamoto (L.) and Leduc (G.) “An Agent-Inspired Active Network Resource Trading Model Applied to Congestion Control”. *2nd International Workshop, MATA'2000*, Paris, France. LNAI 1931, pp 151-169, September 2000.

[51]. An introduction to multiagent systems. Wiley.

[52] http://www.agentbuilder.com

[52] www.swarm.org

[54] http://www.enib.fr/~harrouet/

[55] http://www.madkit.org

[56] http://labs.bt.com/projects/agents/zeus/

[57] http://www.aglets.sourceforge.net

[58] www.recursionsw.com/voyager.htm