UNIVERSIDAD NACIONAL DE COLOMBIA

Fault management in TELCO platforms using Autonomic Computing and Mobile Agents

Sergio Armando Gutiérrez Betancur

Universidad Nacional de Colombia
Facultad de Minas, Escuela de Sistemas
Medellín, Colombia
2011

Fault management in TELCO platforms using Autonomic Computing and Mobile Agents

Sergio Armando Gutiérrez Betancur

Tesis presentada como requisito parcial para optar al título de:
Magister en Ingeniería de Sistemas

Director:
John Willian Branch Bedoya, PhD

Línea de Investigación:
Telecomunicaciones

Universidad Nacional de Colombia
Facultad de Minas, Escuela de Sistemas
Medellin, Colombia
2011

Abstract

This work presents a solution for Fault Management focused in improving the traditional techniques which are used for this purpose in TELCO Service Platforms. This solution is based on two novel technological concepts, as Autonomic Computing, and Mobile Agents.
By means of the use of these two technologies, and taking advantage of their sophisticated features, a solution offering increased availability, proactive reaction to possible failures, faster response times to faults, and reduced bandwidth consumption is developed. The architecture of this solution is discused in detail and cases showing its behavior and performance are presented and discused.

Keywords: Mobile agents, autonomic computing, JADE, Fault Management, APOC

Resumen

Este trabajo presenta una solución para la gestión de fallas enfocada en mejorar las técnicas tradicionales que se usan para este propósito en plataformas de servicio de telecomunicaciones. Esta solución se basa en dos conceptos tecnológicos novedosos, la computación autónoma y los agentes móviles. Mediante estas dos tecnologías, y aprovechando sus características sofisticadas, se desarrolla una solución que ofrece disponibilidad incrementada, reacción practiva a posibles fallas, tiempos de respuestas más rápidos y consumo de ancho de banda reducido. Se discute en detalle la arquitectura de la solución, y se presentan y se discuten casos mostrando su comportamiento y su desempeño.

Palabras clave: Computación autónoma, agentes móviles, gestión de fallas, JADE, APOC.

Content

1 Introduction

The topic of Fault Management has gotten a very high importance, because of its inherent relevance to gain the goal of high availabitility in systems. This topic has a high interest in enterprise and academic communities, and it has become a hot research topic.
This thesis proposes a novel approach for Fault Management, in particular applied to Telecommunications platform domain. This approach consists in an architecture based on Mobile Agents and the principles of Autonomic Computing, which improves the Fault Management process itself by providing:

- Increased accuracy, as the data for the fault detection can be collected in smaller time intervals.
- Learning capabilities, to predict and prevent certain failure conditions.
- Self-correction of failures, by providing intervention mechanisms on the systems components being monitored.
- Increased scalability and stability, by its distributed and mobile nature.

.
This section will introduce some of the reasons which have motivated researching on this topic and attempting to provide an improvement for current methods.
A first element which is important to be mentioned is the fact, that usually system elements, and therefore service platforms do not have Fault Management at design time; Fault Management is added *a-posteriori* and this is a condition which makes very difficult to include this feature. Components and platforms are not designed by nature to be managed. This implies a great challenge, as Fault Management features which be added to system components should adapt to the nature of component themselves, but not being intrusive so that they do not affect the correct function.
Another element, which will be presented in detail in chapter 3, is the fact that most of the methods actually used in Fault Management are based on a Client/Server approach. Client/Server has two main disadvantages, in relation with modern platforms and architectures.

- The trend in system architectures is mainly pointing towards distribution of system components. Proposals as Distributed Service Oriented Architecture [Li and Wu, 2009] and its core concept, Service Delivery Platform (SDP) [Callaway et al., 2008] encourage the distribution or the access of remote components to access services and features. Distribution of services components increases complexity of Fault Management, and makes the Client/Server approach non suitable because of high delays in failure detection, and high bandwidth consumptions [Al-Kasassbeh and Adda, 2008]. Figures **1-1** and **1-2** illustrate these situations and compare them in Client/Server approach and Multi Agent approach.

- In traditional approaches, the server component (named Network Management Station, NMS in the SNMP architecture) only queries the data in management components, and receives alarms, but it usually does not execute any active action in response to the failure. This action has to be performed by a human operator, after acknowledging the alarm or notified event. In few words, current and traditional mechanisms lack of the capacity to correct the failures, and to learn how to avoid them.

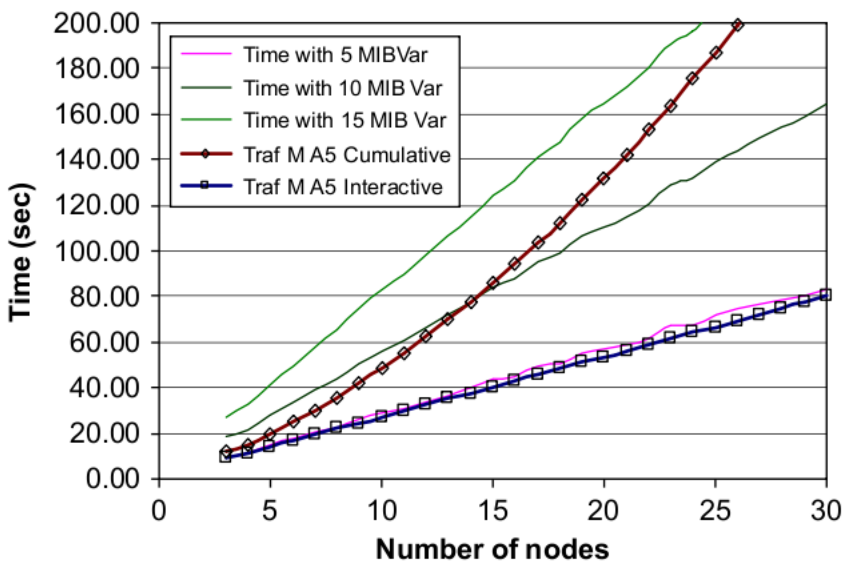

Figure **1-1**: Reaction time when increasing node numbers in traditional SNMP approach and multiagent approach [Al-Kasassbeh and Adda, 2008].

In particular, this last element, more than a disadvantage is a restriction of Client/Server approaches. There exist propietary solutions, as for example Predective Self Healing by Sun Microsystems [Sun Microsystems INC, 2010], which have autonomic features regarding Fault Management, but it is not the general case.
Also, the Client/Server approach for Fault Management, is usually limited to a database which contains the information to be queried and monitored. (In SNMP architecture this is the Management Information Base). Usually, this MIB does not define thresholds which allow to identify failures; these thresholds have to be preconfigured by a human operator. Client/Server approach lacks of self-learning which helps to ease the predictive and proactive monitoring.

1.1 Statement of the problem

Information technologies, and particularly services based or working on Internet have presented a very accelerated evolution in recent years. From the times of Internet dawn to nowadays, the role of these services and technologies has become more important and critical, not only for particular customers, but also for companies whose processes and business goals rely on them.
The evolution in technologies themselves, has caused that multiple networks arise to support them;

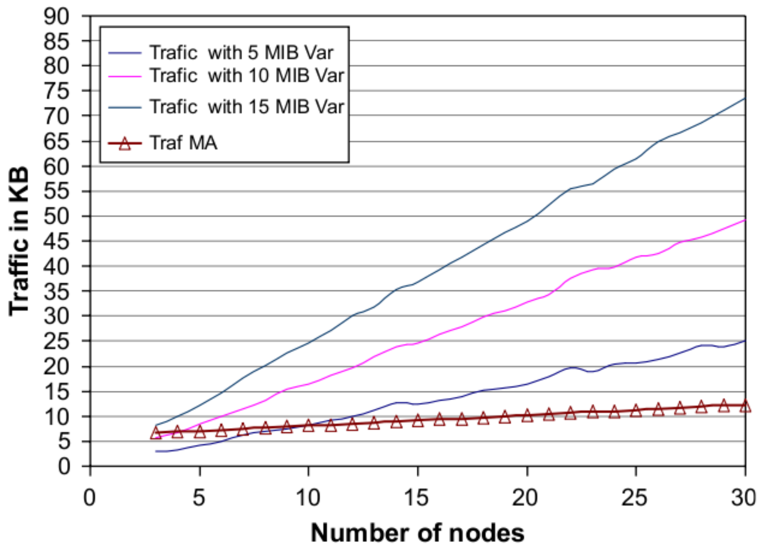

Figure **1-2**: Traffic generation when increasing node numbers in traditional SNMP approach and multiagent approach [Al-Kasassbeh and Adda, 2008].

in a first instance, these networks were different and highly specialized, almost a unique network for every service. Nowadays. the industry trend is offering a single network which be able to support multiple kind of services, giving the adequate treatment for every particular service. This trend is what is known as Service Convergence, or simply Convergence.
As service converge, many different elements of infrastructure are involved in their proper functioning. The increasing complexity related to the relationships among these elements, which on their turn might be located in a very distributed and scattered deployment, and the usually *real-time* requirements, particularly the response to failure events during operation of them, is causing that supervision, monitoring and management complexity of platforms increases in a very dramatic way. A point is being reached, where there will not exist enough qualified personel to support the infrastructure where critical services depend of [IBM Corp., 2005].
Infrastructures where these services run on are considered *critical-mission* because of the high requirements of reliability and availability [Fujisaki et al., 1997]; This is the usually know as *five nines* paradigm.
The capacity of detecting failures in a given system is a very important challenge arising in many engineering disciplines. When dealing with expensive equipments, large and complex systems, or infrastructure where critical services or processes depend on, requirements of availability and reliability increase in a dramatical way. This is a reason that makes Fault Management, becomes a very important research topic in areas as chemical, aerospacial, nuclear and electrical engineering, among many others [Hwang et al., 2010].
One of the most important areas on Telco Network Management, which is directly involved in gaining high levels of availability in platforms is Fault Management (Fault Management). In fact, the most widely used and studied models of IT management and operation define it as a core area [ISO, a][ISO, b][ITI,][ITSMF, 2003]

Fault Management in general, and in particular in a telecommunications platform is the process of determining and correcting problems that arise in the communications network. Typically, when a problem arises in the network (e.g., a device malfunction), it generates a signal (alarm) that is captured by monitoring systems. One difficulty is that the "primary" network problem generally causes connected or neighboring elements to generate "secondary" alarms; for example, from elements expecting confirmation of messages [Kerschberg et al., 1991].
Fault Management can be conceptually divided in three phases: Detection, diagnosis and repairing [V. Baggiolini, 1998].
Detection is based on the analysis of both alarm events generated by platform components, indicating ongoing malfunctions or buggy behaviors (Passive Monitoring) and injection of test signaling or test transactions into system components to assess the answers produced by them (Active Monitoring) [Miller and Arisha, 2001].
Diagnosis is the attempt to detect the root cause or causes of a problem. This diagnosis can be performed by event correlation [Sterritt et al., 2003] or by specializing the probes sent to a particular equipment detected or suspected as failing [Brodie et al., 2002, Chen, 2006].
Repairing is the intervention performed on the faulty component, executing the repairing itself, or removing or isolating it to avoid the overall impact on the whole platform [Sterritt and Bustard, 2002].
Both of the approaches (active or passive monitoring) are usually based on Client/Server (CS) computing model, where there is a centralized entity performing detection and diagnosis, either by analyzing and correlationing the messages sent by components, or by sending and receiving back the testing probes and processing them offline. Traditional approaches to Fault Management do not have inherent mechanisms to perform repairing; This is left to an external human operator or it is performed by a propietary and specific entity into the faulty equipment. CS model presents serious limitation when facing the challenges generated by the evolution of service platforms, which everyday become more and more complex. These limitations cause that detection and correction might not be efficient, accurate and agile enough for service availabiliy requirements. Also, the fact of having a centralized component for fault management, makes itself a single point of failure in the platform. To be effective, a Fault Management system should have an availability and robustness even superior to the platform which is managing itself [Sun et al., 2003].
It is important at this point of problem contextualization, to give a precise definition of some core concepts which are recurrent and very imporant for well understand the problem terminology.
Following to Steinder and Sethi [Steinder and Sethi, 2004], these are the definitions of four core concepts related to fault management:

- *Event*, defined as an exceptional condition occurring in the operation of hardware or software of a managed network, is a central concept pertaining to fault diagnosis.
- *Faults* (also referred to as problems or root causes) constitute a class of network events that can cause other events but are not themselves caused by other events. They can be classified in intermitent, permanent or transient.
- *Error* is defined as a discrepancy between a computed, observed, or measured value or condition and a true, specified, or theoretically correct value or condition. Error is a consequence of a fault. Faults may or may not cause one or more errors.

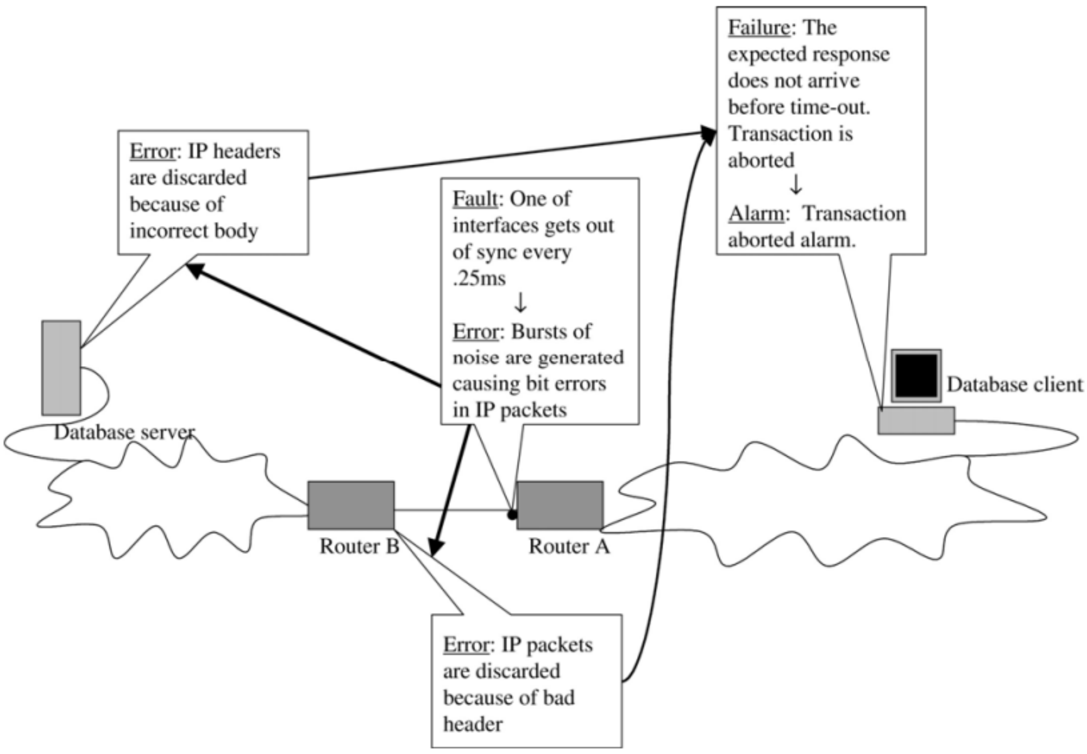
Failure: The expected response does not arrive before time-out. Transaction is aborted
↓
Alarm: Transaction aborted alarm.
Error: IP headers are discarded because of incorrect body
Fault: One of interfaces gets out of sync every .25ms
↓
Error: Bursts of noise are generated causing bit errors in IP packets
Database server
Database client
Router B
Router A
Error: IP packets are discarded because of bad header

Figure **1-3**: Core concepts in Fault Management [Steinder and Sethi, 2004].

- *Symptoms* are external manifestations of failures. They are observed as alarms, which might be notifications of inminent or ongoing failures.

Figure **1-3** illustrates the relationship among core concepts in an example of a real world topology.

1.2 Objectives

1.2.1 General Objective

Design an architecture for Fault Management in TELCO Platforms by implementing the concepts of Autonomic Computing, by means of Mobile Agents.

1.2.2 Specific Objectives

- Compare Fault Management based on Autonomic Computing against a Traditional Fault Management approach by showing advantages and disadvantages.

- Propose an architecture based on Mobile Agents for Fault Management, which shows the features of Autonomic Computing Paradigm, and how they improve Fault Management processes for TELCO platforms.

- Validate the proposed architecture by applying to a real world TELCO platform, evaluating its performance and accurateness in contrast to other methods.
- Compare different mechanisms to access system resources, to provide a non instrusive way to gather information from them for management and monitoring tasks.

1.3 Contribution

In this work, the solution to be presented is an architecture for improving the tasks related to Fault Management, by using mobile agents implementing the features of autonomic computing, in such a way that Fault Management be more agile, more accurate, and more than just detect and report failures, be capable to learn how to correct and then avoid them.

1.4 Organization

This work is organized in the following way: Chapter 2 presents the backgroumd of the core concepts used in the formulation of the architecture being the main contribution of this thesis. Chapter 3 presents a review of literature on the concepts of Autonomic Computing, Mobile Agents and Fault Management. Chapter 4 outlines the architecture proposed for Fault Management in Telco platforms using the paradigms of Autonomic Computing and Mobile Agents. Chapter 5 describes the experiments performed on a prototyple implementing the proposed architecture, to validate its features and finally Chapter 6 collects the conclusions of this work, and future work to be performed from it.

2 Fundamentals

This chapter presents some concepts and terminology which is useful to understand the proposed solution for Fault Management problem, by using Autonomic Computing and Mobile Agents. First, the concept of Fault Management in Telecommunications is introduced. Then, the concepts of Autonomic Computing is introduced and Mobile Agents technology are presented.

2.1 Fault Management.

Fault management (Fault Management) is the collection and analysis of alarms and faults in the service. These faults can be either transient or persistent. Transient failures are not alarmed if their occurrence does not exceed a threshold; for example, sporadic message losses or delays. These events are, however, logged. Some transient problems can be automatically corrected within the service, while others may require different levels of management services to resolve. Faults can be determined from unsolicited alarm messages or by log analysis; the latter may be the only course when, say, existing services/applications do not have internal monitoring and/or alarm generation capabilities. The Fault Management function analyzes and filters the fault messages and coordinates the messages so that the number of actual events reflects the real conditions of the services. The root cause is reported, while suppressing other related fault messages [Goyal et al., 2009].
Several organizations and institutions have formulated diverse approaches to fault management.
The International Organization for Standardization's (ISO) has defined the Network Management Framework. This framework is a reference model created *«[...] to provide a common basis for the coordinated development of management standards [ISO, a]»*. It consists of five conceptual areas that (1) define terminology, (2) create structure, and (3) describe activities for the management of net-centric systems. In particular, Fault Management is defined as an area encompassing fault detection, isolation and the correction of abnormal operation of the Open Systems Interconnection Environment (OSIE). Functionality of Fault Management includes [Gorod et al., 2007] :

- Maintain and examine error logs;
- Accept and act upon error detection notifications;
- Trace and identify faults;
- Carry out sequences of diagnostic tests.
- Correct faults.

In more recent times, organizations have implemented fully standarized frameworks which help to formalize their technology processes. In particular, the two more widely adopted frameworks are ITIL and eTOM, this last one, mainly in TELCO companies.

The Information Technology Infrastructure Library (ITIL) [ITI,] consists of an interrelated set of best practices and processes for lowering the cost, while improving the quality of IT services delivered to users. It is organized around five key domains: business perspective, application management, service delivery, service support, and infrastructure management. ITIL has gained, of all approaches, the biggest popularity and can now indeed be called a *de-facto* standard. Even standards as ISO 20000 [ISO, b], which defines best practices for Service Managament are based on ITIL.
Within ITIL. there are some very important tasks which link Service Delivery and Service Support domains.
As defined in ITIL, Service Level Management ensures continual identification, monitoring and reviewing of the optimally agreed levels of IT services as required by the business. Most targets set in a Service Level Agreement (SLA) are subject to direct financial penalties or indirect financial repercussions if not met. It is therefore critical for this management process to flag when service levels are projected to be violated in order for an IT organization to take proactive actions to address the issue . To this extent, ITIL defines an incident as a deviation from the (expected) standard operation of a system or a service. The objective of Incident Management is to provide continuity by restoring the service in the quickest way possible by whatever means necessary (temporary fixes or workarounds).
As mentioned, ITIL understands the management of service problems (Incidents), more in relationship with the affectation they cause in SLA's and its impact in customer and final user relationship. The work by Salle and Bartollini presents a deeper discussion on this topic [Salle and Bartolini, 2004]. (See Figure **2-1**)

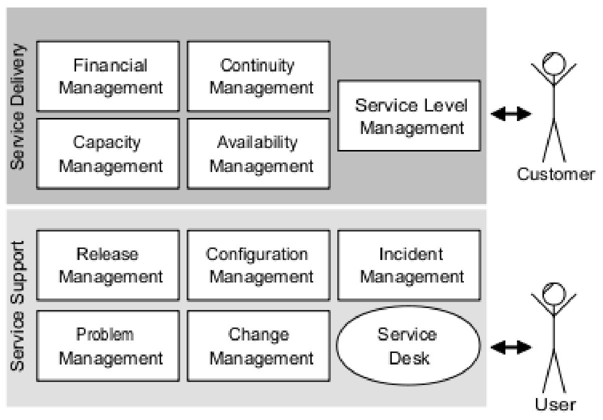

Figure **2-1**: Scope of ITIL IT Service Management [Brenner, 2006]

Recently, and with the goal to adapt itself to Network Evolution and increased growing, The Telecommunications Managament Forum. which is the organization which traditionally has been followed by service providers in the concepts related to service managament, has released the "Enhanced Telecommunications Operation Map", best known as eTOM [TMF,].
The eTOM model (Figure **2-2**) divides the telecom network operations into three main areas: Service Fulfillment, Service Assurance and Billing, of which Service Management is one of the key functions across all domains. Within Service Management, there are various applications such as

Service Problem Management and Service Quality Analysis, Action and Reporting that are closely related to customer services. In this model, service management focus is not to wait passively for customer complaints when there is a service-impacting problem, instead the focus is to identity quickly the customers whose service is being impacted and proactively notify them.
This is a challenging task because it requires (i) clearly-defined service management processes that link the different applications in the eTOM model, and (ii) complex application systems to automate these processes.

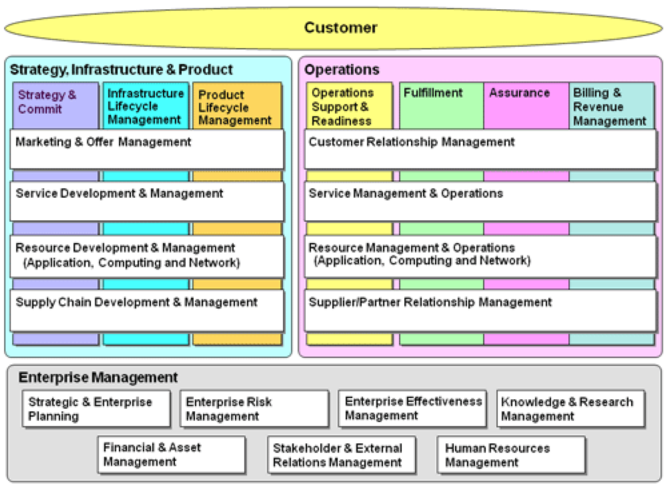
Customer
Strategy, Infrastructure & Product
Strategy & Commit
Infrastructure Lifecycle Management
Product Lifecycle Management
Marketing & Offer Management
Service Development & Management
Resource Development & Management (Application, Computing and Network)
Supply Chain Development & Management
Operations
Operations Support & Readiness
Fulfillment
Assurance
Billing & Revenue Management
Customer Relationship Management
Service Management & Operations
Resource Management & Operations (Application, Computing and Network)
Supplier/Partner Relationship Management
Enterprise Management
Strategic & Enterprise Planning
Enterprise Risk Management
Enterprise Effectiveness Management
Knowledge & Research Management
Financial & Asset Management
Stakeholder & External Relations Management
Human Resources Management

Figure **2-2**: The eTOM Model [Wong, 2009]

Figure **2-3** illustrates an example of Service Management process.
In this example, when the «Network Maintenance» application detects a network element alarm, it immediately corresponds with the «Network Inventory Management» application to check the services that are being impacted by this problem and the affected customers. This information is then passed to the «Service Problem Management» application to see how this service problem should be handled. On the one hand, the «Problem Handling» application would be triggered to inform the affected customers about the service problem. At the same time, the «Customer QoS Management» application will be invoked to verify whether any QoS commitments to the affected customers are being violated. If so, penalty information must be sent to the «Invoice» application to give appropriate credits to the affected customers. Depending on the service offerings and the customer requirements, the service management process for each service impacting problem might be highly complex [Wong, 2009].

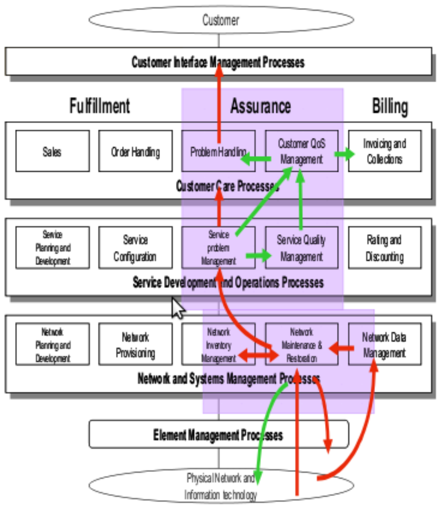
Customer
Customer Interface Management Processes
Fulfillment
Assurance
Billing
Sales
Order Handling
Problem Handling
Customer QoS Management
Invoicing and Collections
Customer Care Processes
Service Planning and Development
Service Configuration
Service problem Management
Service Quality Management
Rating and Discounting
Service Development and Operations Processes
Network Planning and Development
Network Provisioning
Network Inventory Management
Network Maintenance & Restoration
Network Data Management
Network and Systems Management Processes
Element Management Processes
Physical Network and Information technology

Figure **2-3**: Service Management Process Example, according to eTOM [Wong, 2009]

Another relevant framework in the field of Fault Management is FCAPS. FCAPS is the ISO Telecommunications Management Network model and framework for network management. FCAPS is an acronym for Fault, Configuration, Accounting, Performance, Security, the management categories into which the ISO model defines network management tasks. In non billing organizations Accounting is sometimes replaced with Administration.
A comparison between the ITIL IT Service Management and FCAPS would show that FCAPS has most of the critical capabilities necessary for manageability and operability; ITIL of course has lot more in the form of guidance on processes and methods [Goyal et al., 2009].

2.2 Autonomic Computing.

As a solution for the increasing complexity in operation and maintenance of service platforms, appears Autonomic Computing.
Autonomic Computing is a concept inherited from bio-inspired computing in its beginnings. The first work where Autonomy Oriented Computing is presented is the work by Liu *et al* [Liu et al., 2001]. This work defines the fundamentals of Autonomy in computing and the four main characteristics which should be exhibited by a computing system according to this paradigm:

- **Autonomy:** The entities in the system are *rational* individuals, which are capable to act in an independent way; in other words, the system does not have a central component and manager.
- **Emergent:** When system entities cooperate and work together, they exhibit behaviors not available or not possible to be obtained by individual entities separatedly.
- **Adaptive:** System components are capable to modify their behavior according to changes present in the environment where they operate.

- **Self-organized:** The system componentes are capable to organize themselves to achieve the previous commented behaviors.

Authors also present the several types of Autonomy Oriented Computing according to how autonomy is achieved by the system.
In Jin and Liu [Jin and Liu, 2004], Autonomy Oriented Computing is formally defined, by employing set notation to express the concepts associated: Environment, Computing Entity, State, Behavior, Goal are some of the concepts which are expressed in a formal language.
From a more applied point of view, and again from a bioinspired perspective, Horn [Horn, 2001] presents the approach to Autonomic Computing from perspective of a technology industry leader as IBM.
The first part of this work illustrates some of the problems which are inherent to technological evolution. One of the most relevant is the increasing complexity, which is almost reaching a state where operation of service platforms according to traditional approaches will not be possible. Taking as referent the human autonomous nervous system, the author proposes a new paradigm for service platforms named Autonomic Computing. Following Horn, there are eight keys elements which posseses an Autonomic Computing System:

- ***Self-Awareness:*** This is, the system should know itself. It will need detailed knowledge of its components, current status, ultimate capacity, and all connections with other systems to govern itself. It will need to know the extent of its «owned» resources, those it can borrow or lend, and those that may be shared or should be isolated.

- ***Self-Configuration:*** System configuration or «setup» must occur automatically. Also, the system must modify itself, in such a way that its configuration be the most adequate to cope with environment conditions.

- ***Self-Optimization:*** This is, the system will always try to find other ways to improve its functioning. It will monitor its constituent parts and fine tune workflow to achieve predetermined system goals.

- ***Self-Healing:*** System must be able to recover from events that might cause malfunctioning. The system also must be able to detect problem or potential problems, and according to this, define alternate ways to perform, applying reconfiguration to keep the system working.

- ***Self-Protection:*** Starting from the fact of the potentially agressive and hostile environment where system resides, it must be able to detect, protect and identify potential attacks or vulnerabilities, so that it can protect itself and keep working in a secure and consistent state.

- ***Self-adaptability:*** This is, the system must know its environment, the context which surrounds it when operating, and the other entities cohabitating with it. It must be able to adapt to this environment, and its changing conditions, by reconfiguring itself or optimizing itself.

- ***Openness:*** The components in system must be open to communicate each other, and must be able to work with shared technologies. Propietary solutions are not compatible with Au-

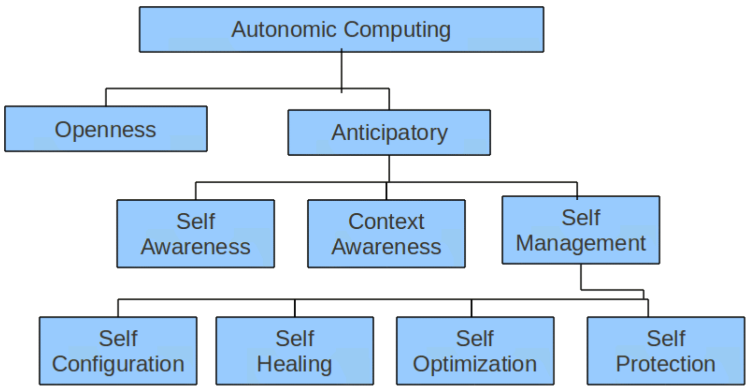

Figure **2-4**: Autonomic Computing Characteristics [Lin et al., 2005]

tonomic Computing philosophy. Components should employ standard protocols, or at least *de facto* standards in their communication.

- ***Self-containment:*** Components within an Autonomic Computing System must be able to perform the task or tasks they have assigned, not requiring external interventions for the performing itself, and hiding the complexity to end user. This does not mean that the system will lack of interfaces allowing the intervention and feedback from human operators.

Complementing the work by Horn, Lin *et al* [Lin et al., 2005] review Autonomic Computing from the perspective of Software Engineering. They present a proposal of metrics which could be used to evaluate the quality of frameworks based on Autonomic Computing.
In other side, the works of Sterritt [R.Sterrit, 2001], Magedanz *et al* [Magedanz, 2004], Ionescu *et al* [Ionescu et al., 2008] and Tizghadam *et al* [Tizghadam and Leon-Garcia, 2010] evidentiate how Autonomic Computing has become a very important research topic in the field of Information Technologies, both for academic communities and for industries, in solving many problems which are becoming difficult to face with traditional and conventional approaches.

2.3 Mobile agents

Previous to introduce the concept of Mobile Agents it is important to establish some concepts related to Agent fundamentals.

2.3.1 Agent fundamentals

Agents are enjoying a lot of popularity as a novel abstraction for structuring distributed applications. Agent approach is a technology from the field of Artificial Intelligence. Agents can deploy distributed applications into a networked information system. Often, they act on behalf of users, enabling task automation and hiding complexity. Software modules, which are autonomous components of software, are equipped with self-contained intelligence capabilities (using a combination of

technologies, including expert systems for reasoning, fuzzy logic for knowledge representation, and machine learning for improving their knowledge) typically implement Agents. They usually operate autonomously, defining and organizing their executions and internal objectives and exerting a certain degree of control over their actions. Moreover, Agents can communicate with user and network resources: they cooperate with other agents to execute tasks that might be beyond a single agent's capability [Boudriga and Obaidat, 2004].
Although, there is not a precise and widely adopted definition for Agents, the tendency is define them through the features they should expose [Franklin and Graesser, 1996]. Following to Yubao and Renyuan [Yubao and Renyuan, 2009], an agent is an entity possesing the following characteristics:

- ***Self-government:*** Agents should have the ability to governate themselves, without external interference from the outside world while they are performing their tasks.
- ***Smart:*** Agents should implement certain functions and be able to choose the required information to complete their tasks. They also should be able to get knowledge from the performing of their tasks.
- ***Lasting:*** The agents should have survival capacity, according to their participation in the tasks.
- ***Co-relation:*** This is the social behavior described by theoretical definitions. In real world, the coperation is presented as messages exchange among the agents in the system.

Jennings and Wooldridge [Jennings and Wooldridge, 1998] define an agent as «... computer system situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives.» An intelligent agent is a computer system that is capable of flexible autonomous action in order to meet its design objectives. By flexible, we mean that the system must be:

- responsive: agents should perceive their environment (which may be the physical world, a user, a collection of agents, the Internet, etc.) and respond in a timely fashion to changes that occur in it,
- proactive: agents should not simply act in response to their environment, they should be able to exhibit opportunistic, goal-directed behavior and take the initiative where appropriate, and
- social: agents should be able to interact, when they deem appropriate, with other artificial agents and humans in order to complete their own problem solving and to help others with their activities.

In Tosic and Agha [Tosic and Agha, 2004], a taxonomy of agents is presented. This hierarchical taxonomy is important because complements the conventional definitions by presenting the neccesary attributes which should expose an agent.
The first of these attributes is *resposiveness* [Wooldridge and Jennings, 1995]. The responsiveness means that the agents have to be able to notice changes in environment, respond to these changes and affect their input or modify their internal state according to these changes.

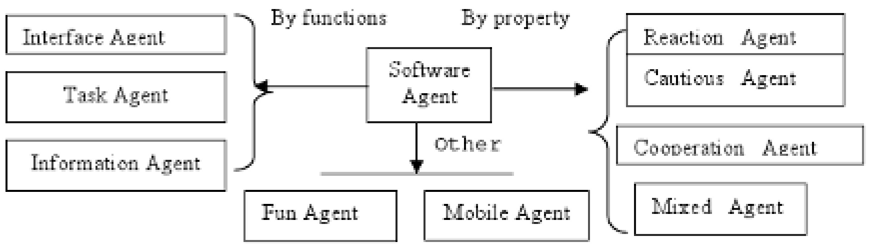

Figure **2-5**: Software Agent Classification [Yubao and Renyuan, 2009]

Another attribute is the *persistence*; this is, the agents should be *alive* on their own while they perform their task. This is a behavior basically different than the exhibited by a subroutine within a program, which is turned on or off on demand.
The *goal-orientedness* or *goal-drivenness* is another important characteristic of agents. In computational agents, it is related to the fact that the agent is implemented to perform a specific task or to solve a very defined problem, so, the goal during the execution of the agent is perform that task or solve that problem.
Finally, the last attribute which is presented by the author is *proactiveness.* This attribute refers to the fact that agent is not executing by cycling randomly around its internal states, but in a way which allows it to get its goal.
Figure **2-5** presents a schematic classification of software agents.
In other hand, Boudriga and Obaidat [Boudriga and Obaidat, 2004] define the main capabilities to be exposed by a system based on Agents:

- Intelligence. The method an agent uses to develop its intelligence includes using the agent's own software content and knowledge representation, which describes vocabulary data, conditions, goals, and tasks.

- Continuity. An agent is a continuously running process that can detect changes in its environment, modify its behavior, and update its knowledge base (which describes the environment). It can be triggered by the occurrence of events, a time condition, or environmental constraints. Continuity defines how the agent is triggered and with what degree of synchronization it operates.

- Communication. An agent can communicate with other agents to achieve its goals, and it can interact with users directly by using appropriate interfaces. To do so, a wide range of resources and protocol services can be accessed. An agent's communication attribute describes resources, protocols, and services with which the agent is allowed to interoperate.

- Cooperation. An agent automatically customizes itself to its users' needs based on previous experiences and monitored profiles. It can also automatically adapt itself to changes. The interoperation's complexity might have several types: client, server, or negotiations based on various intelligent methods. The cooperation attribute describes the nature of cooperation and the prerequisite for multiagent interoperation.

- Mobility. The degree of mobility with which an agent can perform varies from remote execution, in which the agent is transferred from a distant system, to a situation in which the agent creates new agents, dies, or executes partially during migration.

2.3.2 Mobile Agents

Although the basics of Agents provide some support for distribution, the current evolution of technologies and architectures, where system components are highly distributed and even system composition is dynamic (new components enter and leave during system performing) consitute a challenge in design and development of service platforms. The emergence of Wireless and Cellular networks rehearse this challenge. As an approach to this challenge appear solutions focused in logical mobility, code and state mobility of the computational entities. The so named mobile agents implement a variation of this behavior, where they can be migrated in state and code to different locations, and it is the agent itself who autonomously decides about this migration [Picco, 2001].
Mobile agents are such agents that can move in a computer network from host to host as needed in accomplishing their tasks. Mobile agents offer a natural extension of the remote programming (RP) paradigm in several interesting ways. In particular, mobile agents are generally thought to be able to act with a certain degree of autonomy. That is, the agent is able to make intelligent decisions regarding its itinerary and modify it in a dynamic fashion in response to information that becomes available as it moves from one host to another [Yang et al., 1998].
Mobile agents provide a potentially efficient framework for performing computation in a distributed fashion at sites where the relevant data is available instead of expensive shipping of large volumes of data across the network. For example, in many data mining and knowledge discovery tasks, a mobile agent can visit multiple, geographically distributed, data repositories and return with knowledge.
They are also defined as programs that may be dispatched from a computer to another for execution and interaction with other agents. They are not bound to the system where they begin execution, but they can transport themselves to different hosts in the network. Thus, the hosts become very flexible, because the computation can be performed on any host by means of the agents, which can move form one host to another in order to get the resources and computing power they need to complete their tasks [Zaharia et al., 2003].
Zaharia *et al* [Zaharia et al., 2003] gather seven reasons which highlight the use mobile agents as paradigm for implementing distributed applications:

- They reduce the network load by reducing raw data flow over network;
- They overcome network latency in real-time systems;
- They encapsulate protocols, i.e. establish channels for communication;
- Asynchronous and autonomous execution;
- Dynamic adaptation (they sense their environment and they are able to adapt to changes);
- Heterogeneity (they depend only on their execution environments);
- They are robust and fault tolerant.

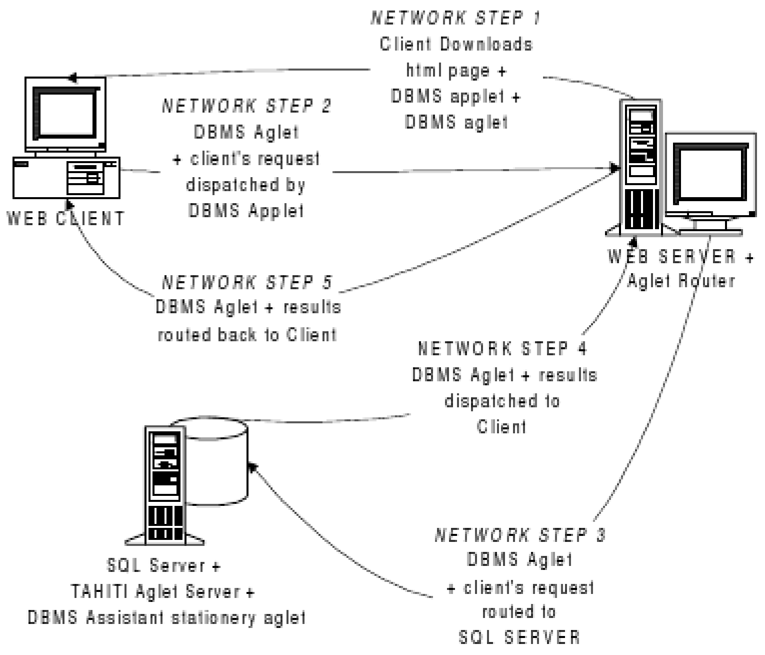
NETWORK STEP 1
Client Downloads html page + DBMS applet + DBMS aglet
NETWORK STEP 2
DBMS Aglet + client's request dispatched by DBMS Applet
WEB CLIENT
WEB SERVER + Aglet Router
NETWORK STEP 5
DBMS Aglet + results routed back to Client
NETWORK STEP 4
DBMS Aglet + results dispatched to Client
SQL Server + TAHITI Aglet Server + DBMS Assistant stationery aglet
NETWORK STEP 3
DBMS Aglet + client's request routed to SQL SERVER

Figure **2-6**: Mobile Agents in Distributed Database Access [Samaras et al., 1999].

The important key contribution of agent mobility is the fact that the execution of the agent is strongly decoupled of its location; It could be understood as the agent is where it is really needed, and even the agent could decide to migrate itself to locations with more resources for fullfiling its goal. This perspective offers multiple possibilites for the evolution of distributed computing and applications. Agent platform could be implemented in such a way that provides independence of operating system and hardware platform.
Although there are authors which pretended to reduce the applicability of Mobile Agents (See Harrison *et al* [Harrison et al., 1995]), they can be considered as another approach to be used for the design and implementation of distributed applications, and which fits very adequately to the *state-of-art* of modern technologies and architectures.
Picco [Picco, 2001] in his introduction to mobile agents illustrates two success cases in applying Mobile Agents to solve the problem of reducing Database query times [Samaras et al., 1999] (Figure **2-6**) and Data collection for Network Management Systems [Baldi and Picco, 1998]. Other works in this area are the works by Adhicandra *et al* [Adhicandra et al., 2003] and Gavalas *et al* [Gavalas et al., 2009] (Figure **2-7**)
These experiencies in applying Mobile Agents in Network Management will be an important starting point for the work which will be presented on this thesis.

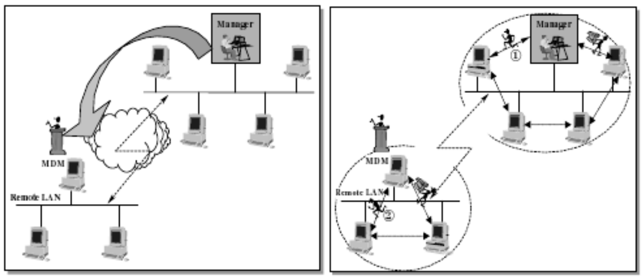

Figure **2-7**: Mobile Agents in Network Management [Gavalas et al., 2009]

3 State of Art and related works

3.1 Introduction

This chapter will present works related to Fault Management. As it will be shown, the existing approaches (except by Self-Maintenance) are based in Client/Server model, and lack of mechanisms for repairing the faulty components. Also, they lack of mechanisms which allow to anticipate to fault conditions, preventing them in a predictive or proactive way.
The first step in fault management is fault detection, which is usually performed via System Monitoring.
Bagglioni and Harms [V. Baggiolini, 1998] mention two main features which are common to most of Fault Management approaches and techniques:

- They are generally added *a-posteriori* to existing applications. This means that applications are not designed for being managed. They often lack a suitable architecture and efficacious means for diagnosis and repair of faults.

- They typically use external management functionality. State or behavior information is extracted from the application and analyzed by an external manager. This means that problems are torn out of their live context, which (1) makes them much more difficult to diagnose and correct, and (2) breaks encapsulation. This defeats good software engineering principles, hinders reusability, and does not favor automatic management solutions.

Traditional and conventional Fault Management systems are based on a Client/Server (C/S) model [Al-Kasassbeh and Adda, 2008], where alarms and events are captured by sensors in system components, and sent them to a centralized entity, the Management Server or Management Station, where they are collected, and notifications are sent to system operators so that they perform the diagnosis and correction of the faulty component and situation (Figure **3-1**).
Other common scenario is the information collection, which is triggered and performed from the management entity by querying the managed devices. Further analysis is performed by external applications. This is the main working principle of Simple Network Management Protocol (SNMP) [Harrington et al., 2002], which is currently the most used basis in Fault Management systems
Serious limitations of this traditional approach are the requirement of human intervention for fault diagnosis and correction, and the lack of automatic learning from the failure occurrence; in this approach, the knowledge has to be acquired, documented and stored by human system operators and administrators, and the application of this knowledge is upon these same human actors.
Also, from data which is periodically gathered from managed device, further analysis has to be performed offline; no knowledge is discovered from these data, and diagnosis has to be performed by human operators.

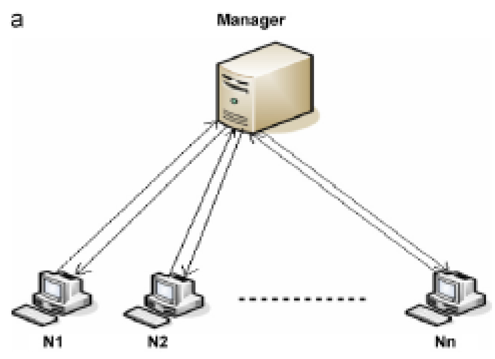

Figure **3-1**: Traditional Fault Management [Al-Kasassbeh and Adda, 2008]

Finally, altough C/S model is very reliable, it has problems regarding scalability and capacity to handle in an adequate way very distributed deployments or high quantity of elements [Pras et al., 2004] and might present high consumptions of bandwidth [Al-Kasassbeh and Adda, 2008].
Zaw and Soe [Zaw and Soe, 2008] present a typical C/S solution for Network Management in Local Area Network environments; this work exhibits the common features of C/S approach to Fault Management.
Wang and Zhou [Wang and Zhou, 2004] present and extension to C/S monitoring based on SNMP [Harrington et al., 2002] for Grid platforms, which solves some of the scalability problems which are inherent to C/S model itself.

3.2 Novel approaches

Maintenance is a task which is performed on systems, and its main purpose is keep or restore the condition of the system to a predefined state. Maintenance can be preventive, when it is performed on a timely basis, to deal with degradation or aging or system parts, or corrective, when it is performed after a failure occurs. There is another novel approach named proactive maintenance, where current and past information and data of system state is processed to detect possible failures yet to occur, even forecasting the time before a failure occurs.

3.2.1 Bio-Inspired approaches

With the goal of offering increased availability in systems, industry has defined the concept of *Self-Maintenance* (SM).
SM is taken from nature, emulating the capacity of self-repairing which is exhibited by some living organisms.
Mei-hui *et al* [Mei-hui et al., 2010] introduce an approach to SM. According to authors, SM is part of a wider feature, *Self-Recovery* (SR). SR can be understood as the combination of *Self-Maintenance* and *Self-Repairing.*
Self-Maintenance is composed by technologies as Recomposition, Dynamic Reconfiguration, Cold Backups, and Reconstruction, that is to say, technologies which allow to modify or even restore the state of the system to fulfill with some service levels.

Self-maintenance is not a feature considered indispensable for the systems; it is more an evolution and improvement of conventional maintenance systems. *Self-maintenance* component is conformed by Detection System, which are the sensors on charge to detect events and issues; the information of these sensors is then transmitted to the Analysis System[Yuniarto and Labib, 2006], which evaluates the situation according to fault type, and then, transmit to Maintenance System the decision about how to perform the maintenance itself.
Figure 3.2 presents a flow diagram which illustrates the concept of Self-Maintenance.

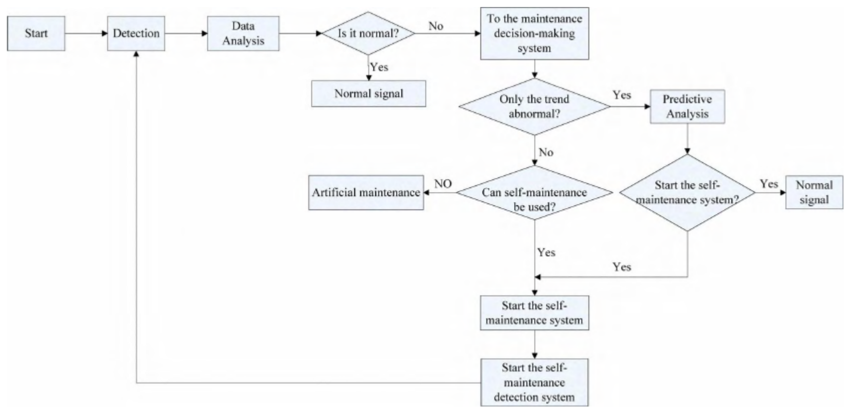
Start
Detection
Data Analysis
Is it normal?
No
Yes
Normal signal
To the maintenance decision-making system
Only the trend abnormal?
Yes
Predictive Analysis
No
Artificial maintenance
NO
Can self-maintenance be used?
Start the self-maintenance system?
Yes
Normal signal
Yes
Yes
Start the self-maintenance system
Start the self-maintenance detection system

Figure **3-2**: The development of Self-Maintenance Process [Mei-hui et al., 2010]

Labib [Labib, 2006] presents the notion of Next Generation Maintenance Systems, as an approach for the design of *Self-Maintenance* machines. In the first part of this work, the generations of maintenance are presented, with a first generation, lasting approximately until World War II, when maintenance was merely corrective, having fault as something unavoidable, so maintenance consisted in fixing equipment, a second generation lasting until 70's decade, when maintenance was focused in offering a high availability on equipments, while costs tried to be reduced. In this second generation, some statistical techniques started to be used in fault modelling, and computer systems, although the ones of these times were quite big and slow. The third generation, which is considered to extend to present time, when maintenance is focused in offering high availability and reliability, but with greater emphasis in environment and security.
So, the fourth generation will be an evolution of third, aiming to zero downtime in systems, by means of *hot redundancies.* It might be that the vision of this generation aims to avoid failure consequences, rather than failures themselves. To gain this, maintenance techniques should combine self-maintaning, self-repair and self-healing features.
Figure **3-3** shows the schematic division of Maintenance Strategies.
This work also presents some unmet needs in responsive maintenance, this is, features which might be considered as desirable in Self-maintenance systems: a) Intelligent monitoring, prediction, prevention and compensation for sustainability, b) Prioritization, Optimization and responsive scheduling for reconfiguration needs and c) Autonomous information flows from market demands to factory asset utilization.
In particular, the last point is very important, as mentioned that most of the maintenance method do not provide feedback to bussiness process within the organization, being these isolated of corporate

policies and merely focused on infrastructure rather than services and processes.
An important difference which is presented between traditional maintenance (preventive and corrective) and Self-Maintenance is the fact that traditional maintenance consists in interventions to avoid more catasthrophic failures, while Self-Maintenance is based in intelligent monitoring, with adaptive and responsing behavior in the system. The features required in the system to be able to implement Self-Maintenance are perception, fault classification and diagnosis, failure prediction, repair planning and repair execution [Sirvunnabood and Labib, 2005].

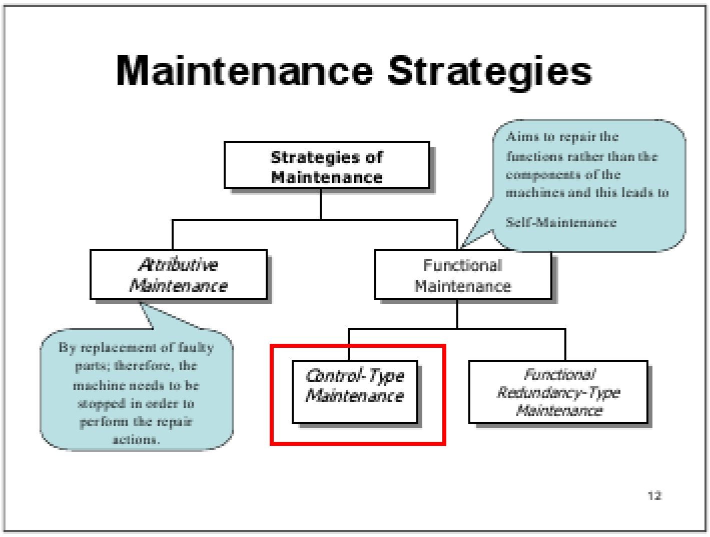
Maintenance Strategies
Strategies of Maintenance
Aims to repair the functions rather than the components of the machines and this leads to Self-Maintenance
Attributive Maintenance
Functional Maintenance
By replacement of faulty parts; therefore, the machine needs to be stopped in order to perform the repair actions.
Control-Type Maintenance
Functional Redundancy-Type Maintenance
12

Figure **3-3**: Maintenance Strategies [Labib, 2006]

Author refers a classification for maintenance strategies [Umeda et al., 1995]: Attributive and Functional maintenance. The attributive maintenance consists in replacing failing parts (which implies stopping the system) while functional maintenance tries to repair the functions rather than components. Functional maintenance focuses in the whole system, and would allow that Self-Maintenance be implemented. Figure **3-3** briefs the classification of maintenance strategies.
Some features wich are required for Self-Maintenance are presented, as follows:

- Monitoring: Self-Maintenance systems must have sensor capabilities; sensors would collect raw data from system components, and this data would be sent to a processing entity.
- Fault judging: The processing entity would take the information coming from sensors and would judge whether system component is on normal or abnormal state. It would be desirable also that from this raw data, the system would try to predict time left for a failure.
- Diagnosing: After analysing the data, whether an abnormal state is found, the system should classify the failure, and identify if possible its root causes, so that a reparing plan can be carried out.
- Repair planning: From the information collected, the system should be capable to propose one or several maintenance plans. These plans would include also information previously stored,

taken from experts. As many repair plans might be proposed, the system should decide to apply an optimized one.

- Repair execution: The action of maintenance which is executed by the system itself. This is performed by control and actuators on the system components.
- Self-Learning and improvement: Although an unexpected problem arises, the system is expected to include the applied solution as knowledge for a possible next time where the same failure arises. This feature will allow the system to repair itself in a shorter time frame, being more efficient and effective.

Sterrit *et al* [Sterritt et al., 2003] introduce the concept of Autonomy in event correlation. Starting from the ideas formulated by Horn [Horn, 2001], authors present an approach where the goal is avoiding failures at system level, by performing self-configuration to ensure minimal disruption. This approach is composed by three main areas:

- Correlation: Consists in processin information which cames from several sources, and trying to find relationships among the information pieces, which be useful to decide when and what to do.
- Rule discovery: The information sent by network entities is just symptoms. A further processing needs to be performed according to rules guiding what events correlate, and how to process the information within them. Techniques as machine learning or data mining might be useful at this point [Sterritt and Bustard, 2002], although in most of cases the most useful knowledge will be the one which could be provided by a human operator [Uthurusamy, 1996].
- Autonomic Computing Correlator Analysis Tools: It is the mechanism which perform the analysis to correlate the events reported, and try to find the root cause of the problem or failure reported. It implements the self-diagnosis, and triggers the self-healing on system components.

Figure **3-4** illustrates the high level design of this last component.

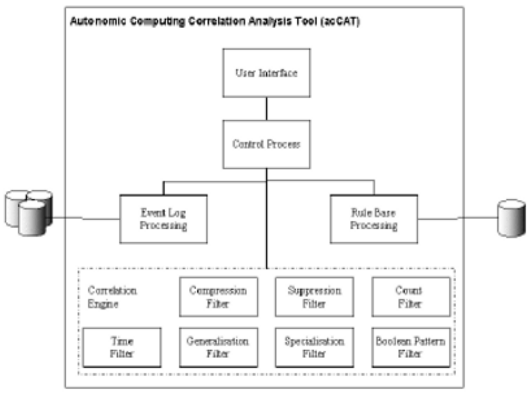

Figure **3-4**: Design of Autonomic Computing Correlator Tool [Sterritt et al., 2003]

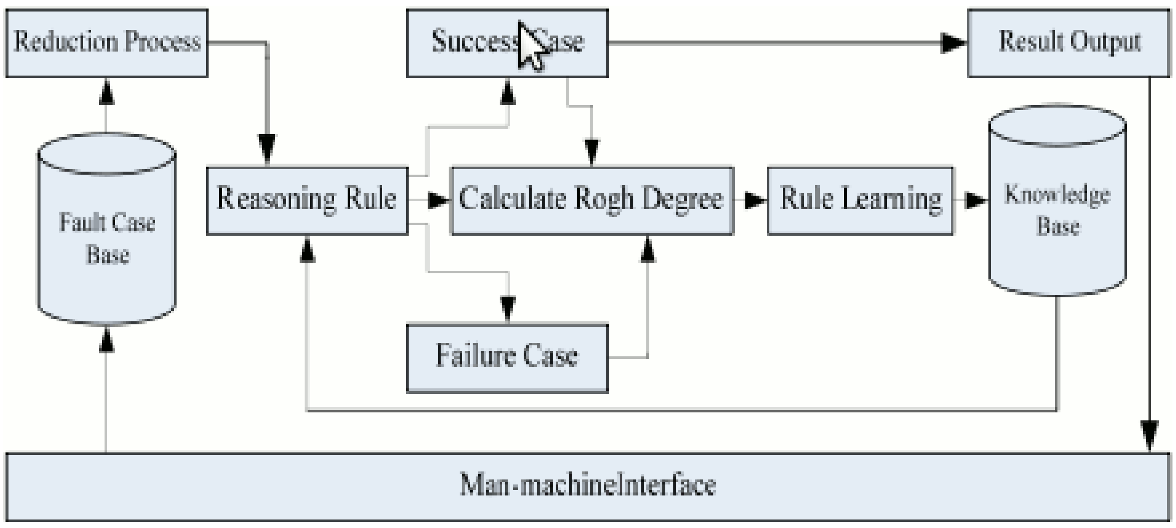

Figure **3-5**: Fault diagnosis with Rough Sets and Expert Systems [Yougang, 2009]

Sterrit [Sterritt, 2001] describe in detail the process of rule discovery by means of acquisition of knowledge from events captured in network and from human expert knowledge, and later applying data mining to this data to obtain new rules for event and alarm correlation. This work opens a novel research topic, as the knowledge acquisition is vital to build monitor and fault management systems which really adapt to changing network conditions and be able to work in real time.

3.2.2 Artificial Intelligence Based

Yougang [Yougang, 2009] proposes an approach based on Rough Sets Theory and Expert Systems. This approach shows an improved diagnosis efficiency by reducing redundant data and the occurrence of false alarms. By being based on Expert Systems, it still has the limitations of conventional Expert Systems, as bottleneck in knowledge acquisition, and dynamic reconfiguration of existing knowledge.
Figure **3-5** illustrates the scheme of the system proposed by Yougang.
Al-Fuqaha *et al* [Al-Fuqaha et al., 2009] present a system called CHARS: Call Home Analysis and Response System. CHARS receives a stream of events, which summarize the state of Software Based Services and Network Elements, and also performs tasks which traditionally were performed by System Operators. CHARS has the ability of correlate the messages produced by services and elements, to detect the root cause of events and issues, and associate them with a solution procedure. This system has the ability of defining relationships and dependencies among service components, and allows to human operators bring knowledge into te system by defined scripts which can react to problems, modifying the configuration of system elements. CHARS has as its core a Rule-Based Expert System.
The main design goals of this system are:

- Intelligence: By using Expert Systems, CHARS has the possibility of store and *forward chaining* the documention and solutions which were previously inserted by human experts, or deduced when solving a particular problem.

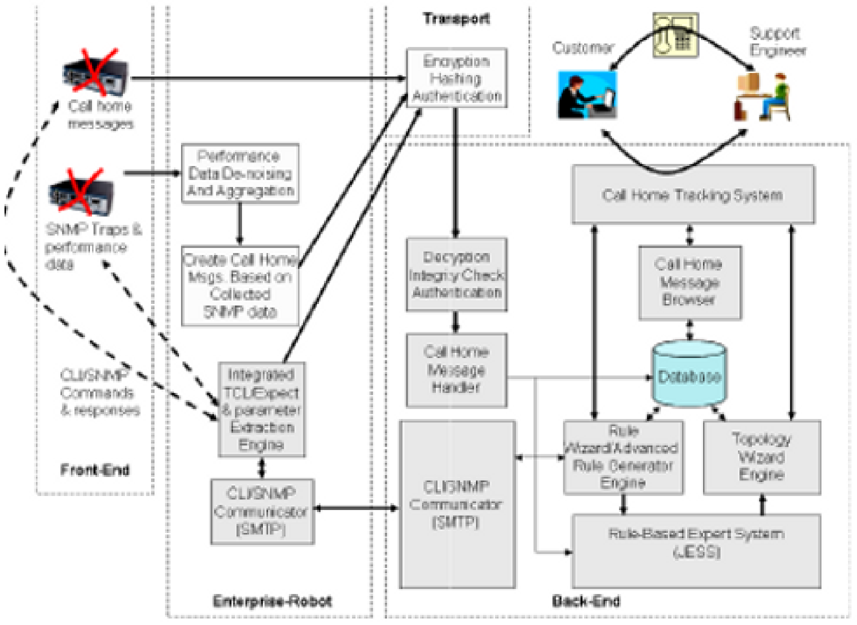
Transport
Customer
Support Engineer
Encryption Hashing Authentication
Call home messages
Performance Data De-noising And Aggregation
SNMP Traps & performance data
Create Call Home Msgs. Based on Collected SNMP data
Call Home Tracking System
Decryption Integrity Check Authentication
Call Home Message Browser
Call Home Message Handler
Database
CLI/SNMP Commands & responses
Integrated TCL/Expect & parameter Extraction Engine
Rule Wizard/Advanced Rule Generator Engine
Topology Wizard Engine
Front-End
CLI/SNMP Communicator (SMTP)
CLI/SNMP Communicator (SMTP)
Rule-Based Expert System (JESS)
Enterprise-Robot
Back-End

Figure **3-6**: Architecture of CHARS [Al-Fuqaha et al., 2009]

- Reactiveness: By using scripts which can interact with network entities, CHARS has the capacity of isolating outages which are detected by *forward chaining*.
- Scalability: As the knowledge of the system is stored in the database of the Expert system, bring new external knowledge into the system is not a hard issue.
- Extensibility: By using technologies as XML, extending the components of the system, for example the rules engine, is quite easy to perform.

Figure **3-6** shows the architecture of CHARS.
Varga and Moldovan [Varga and Moldovan, 2007] propose an Integrated Service Level Monitoring and Fault Management (ISF) framework. This framework offers a solution for service-level monitoring, with the goal of offering end-to-end Quality of Service. This framework is applied in managing several Local Area Networks, thus showing its applicability in multiprovider environments.
A system implemented following this framework, would be integrated by the following elements:

- Event collector: It is an entity where the messages sent by all the nodes and elements in the system send their messages. At this element, they are normalized, stored in the event database, and sent to preprocessing module.
- Data Miner: It is an entity on charge of analyze the event database. It applies analysis algorithms to detect non matching patterns, and in case they are found, alarms are sent to event collector, because they might indicate possible problems in system.
- Event Processing: It is integrated by two submodules, correlator, which tries to find correlations among event messages, and filter, which eliminates unnecesary events.

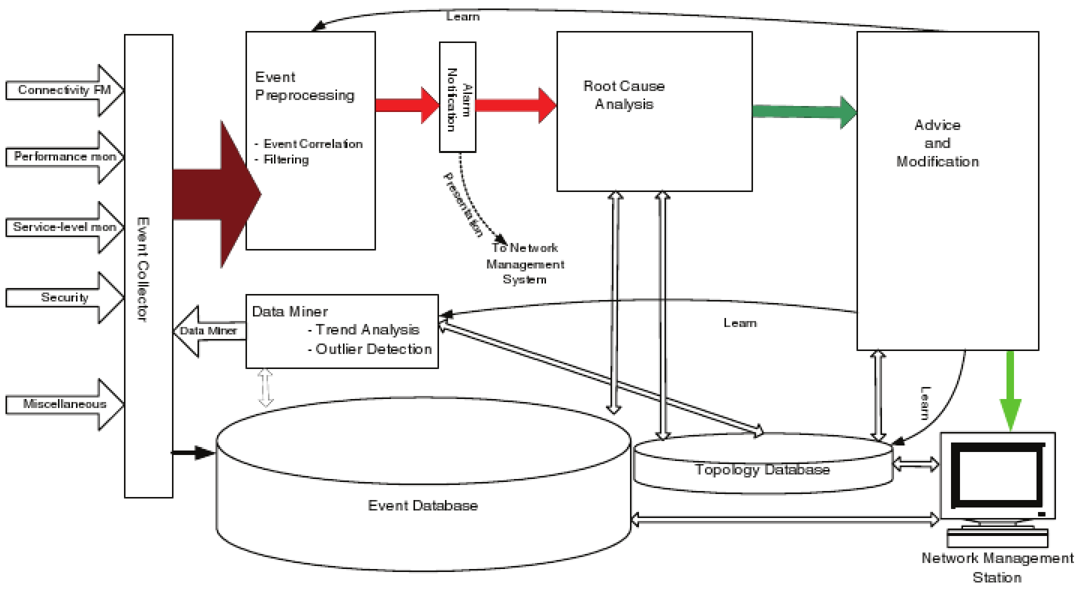

Figure **3-7**: Event processing module [Varga and Moldovan, 2007]

- Alarm Presentation: It is the entity which displays the alarms, and triggers a new Root Cause Analysis for each one.
- Root Cause Analysis: It is an entity which using the descriptive information on the alarm, tries to find a Root Cause for it.
- Advice and Notification: It produces fault descriptions after the Root Cause Analysis. It uses a database to compare the descriptive information and provide possible corrective actions to be taken. This module could just notify, or take directly the actions on the network elements.
- Event Database: It stores the information of the collector, and is used by the data miner for the analysis.
- Topology Database: It stores information of the real network structure. It contains node addresses, node functionalities and connections among nodes.

In particular, for the Root Cause Analysis, this approach uses Petri-Nets as analysis technique as mentioned in [Aghasaryan et al., 1997]. Figure **3-7** shows the event processing module, which is the core and most important component of the proposed framework
Sterrit *et al* [Sterritt et al., 2002] present HACKER, which is a tool which integrates visualization and data mining, incorporating principles of Computer and Human Discovery. The tool is designed to assist in discovering unknown rules from events in such a way that these new rules can be used to feed systems of event correlation based in rules. This process of knowledge discovery is performed in three tiers: Visualization Correlation, Knowledge acquisition (Rule based correlation) and knowledge discovery (Data Mining Correlation). Figure **3-8** shows a diagram illustrating the three tier knowledge discovery process.

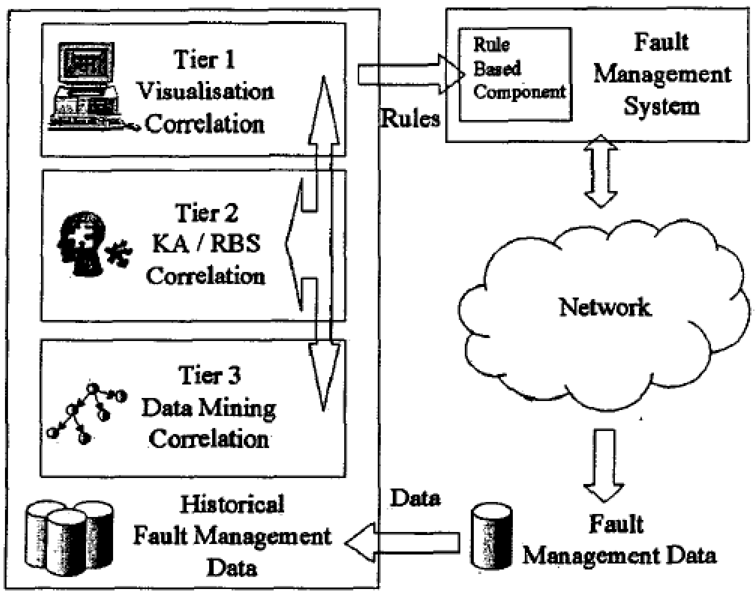

Figure **3-8**: The three tier knowledge discovery process [Sterritt et al., 2002].

3.2.3 Mathematic Techniques Based

Another approach which hase been used for Fault Management and System Monitoring is by means of statistics.

Bhattacharyya *et al* [Bhattacharyya et al., 2004] present the use of discrete events for Fault Diagnosis and Detection. It is based on passive monitoring, and models the network nodes as Finite State Machines, considering faults at Input, Output and Transitions of them. This work has some limitations because, although it is able to detect all faults, it founds some kind of failures which are not diagnosable.

Li *et al* [Li et al., 2009b] propose a method based on Bayesian Network to model the fault propagation on Distributed Application Systems. This approach presents a good behavior, but requires the knowledge of a human expert to increase its performance, so it lacks of autonomy in its working.

Li *et al* [Li et al., 2009a] study the active monitoring of Internet Services as Fault Detection mechanism. This approach tries to solve the problematic related to balance the number of probes which are sent, keeping the balance between enough probes for diagnose, but not increasing it on a measure which could saturate the network. This approach proposes the integration of *a-priori* fault distribution learning, and fault diagnosis in a Hidden Markov Model. The approach as presented, is very accurate to work, even in noisy and uncertain environments, reducing the complexity related to priory knowledge acquisition. Figure **3-9** illustrates the diagnosis probing selection method.

Chu *et al* [Chu et al., 2009] present an approach based in two algorithms, one for probe selection, to monitor the whole components, and other for the fault diagnosis, by sending more probes to obtain a more detailed view of system state. This work focuses in Service Concept, conceiving the service as a composition of interdependent components, in such a way as whether a component fails, it for sure will cause a degradation or failure in the other components. It uses for modelling the

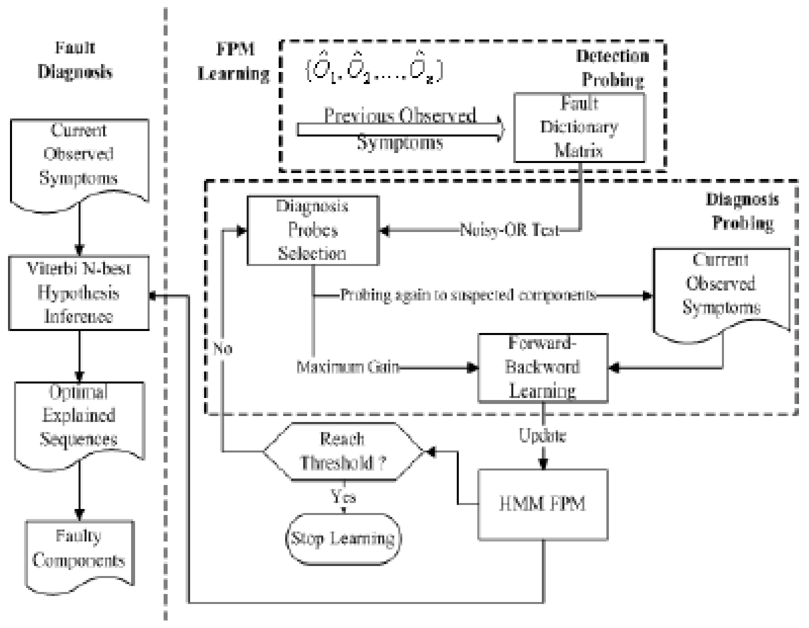
Fault Diagnosis
FPM Learning
$\{\hat{O}_1, \hat{O}_2, \ldots, \hat{O}_z\}$
Detection Probing
Previous Observed Symptoms
Fault Dictionary Matrix
Current Observed Symptoms
Diagnosis Probes Selection
Noisy-OR Test
Diagnosis Probing
Viterbi N-best Hypothesis Inference
Probing again to suspected components
Current Observed Symptoms
No
Maximum Gain
Forward-Backward Learning
Optimal Explained Sequences
Update
Reach Threshold ?
Yes
HMM FPM
Stop Learning
Faulty Components

Figure **3-9**: Diagnosis Probe selection [Li et al., 2009a]

dependencies among components a Bipartite Bayesian Network.
Chu proposes a layer model to understand the service, having the following layers:

- Demand Layer: It comprises the quality of service requirements defined for a particular service. By measuring the service performance, and evaluation to see whether the service is fullfiling these requirements or not. In case that requirements do not be fulfilled, corresponding alarms will be issued and sent to monitoring component.
- Service Layer: In this layer appear service themselves are defined in the system. There exists a *one-to-many* mapping among service and quality of service requirements.
- Dependency Layer: Contains the dependency relationships among system components. This layer applies for the cases where dependency model is adopted to define the system.
- Component Layer: This layer contains the components which integrate a service. In general, each service depends on several components, and different service may share dependent components.
- Mode Layer: This layer defines the modes or states where a service can be located at. One of this modes is defined as working mode, and the other will represent different degrees of degradation.

This work shows an approach that reduces the necessity of probes, but produces a high diagnosis rate, and a low number of false positives.
Figure **3-10** illustrates the representation of dependency model at service layer.

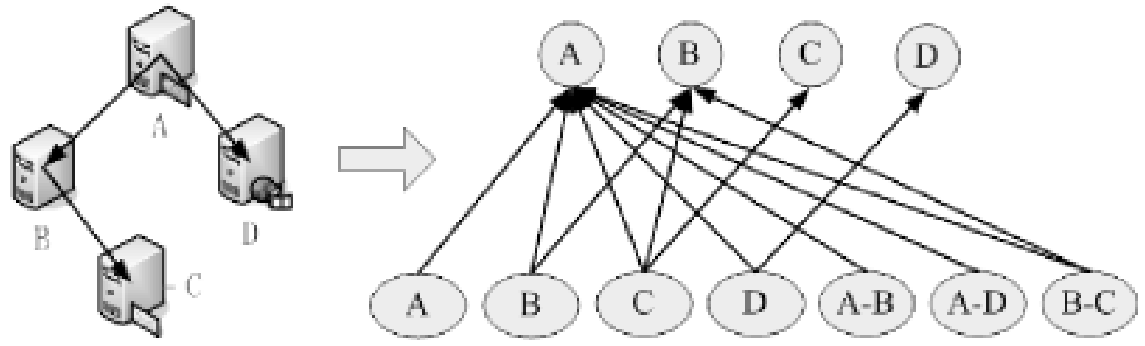

Figure **3-10**: Dependencies model at service layer [Chu et al., 2009]

Chen [Chen, 2005][Chen, 2006] proposes an scheme named Proactive Probing and Probing on demand which can be used to monitor the whole components of a service. He defines an entity called Probing Service (PS), which is implemented as a Web Service, capable of performing a specific transacion, record its result, and send it to a management station. These services are deployed by means of Probing Service Outlets (PSO), where PS can invoked on a regular basis, or can be invoked at anytime.
The scheme proposed by the author is composed by three stages. A first stage, when PSO and PS are selected to monitor service components; the goal is select a set of PS as small as possible. At this point, the information is collected with the only purpose of monitoring the health of the system, and will not be useful for diagnosing the particular problem. The advantage of decoupling monitoring and diagnosing is reducen overhead and traffic, as system will be most of the time monitoring, with, as mentioned, the smallest set of PS.
The second stage is offline rule development, that is to say, provide and strategy in the case of one or several PS reports problems. This strategy should consider the whole possible pools of PS, for the failure detection. The last stage of this approach is enabling On Demand and Incremental probing, for the failure diagnosing and location. It is performed by using one or several PSs to get additional information, and it is guided by the strategy which was defined on second stage.
Figure **3-11** illustrates the generic architecture of probing systems.
Kirmani and Hood [Kirmani and Hood, 2004] propose a mechanism to diagnose network states by means of a technique named Intelligent Probing, presented by Brodie *et al* [Brodie et al., 2002]. Intelligent probing consists in finding an optimal set of probes, which tends to be small but still enough to get an accurate diagnosis, which be efficient in terms of precision and time.

3.3 Contributions of this thesis

After reviewing the *state-of-art* of Fault Management approaches, in particular traditional ones, the following limitations, which are to be dealt with this proposal are:

- Traditional Fault Management approaches are based on client server model where exists a centralized entity which queries and analyzes data from managed entities. This deployment is not suitable for very distributed platforms, or for platforms formed by a high number

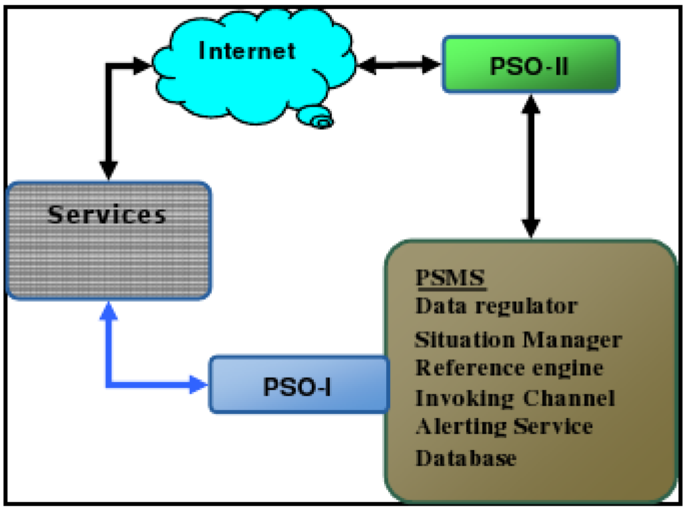

Figure **3-11**: Generic architecture of probing systems [Chen, 2005, Chen, 2006]

of components. Experience shows that when the number of managed nodes increases, the performance of Fault Management system decreases [Al-Kasassbeh and Adda, 2008].

- Traditional Fault Management approaches do not support mobility. Mobility is a feature which is very important in the modern technologies, as it provides that entities migrate through the networks and platforms. An effective Fault Management approach should have mobility features, to be able to detect, diagnose and correct problems in these mobile entities.
- Traditional Fault Management approaches do not include entities capable to solve in an autonomous way abnormal situations which might arise. Usually, the execution of actions tending to solve the failure are triggered or executed by an external entity.
- Conventional Fault Management systems lack of intelligence to perform predictive or proactive maintenance. Situations that in some time could drive to a failure, are not usually identified as such ones, and actions are not taken regarding them.

4 An architecture for fault management in TELCO platforms

This chapter will present the detailed design of the solution to be proposed on this thesis for Fault Management by means of mobile agents and autonomic computing principles.
The chapter is divided in three main sections. The first section will introduce the theoretical support used to define the architectural framework of the design. Second section will introduce the architecture of the design and third section will analyze in detail the features exhibited by each component in the architecture.

4.1 Reference framework.

In the work of design a system based on the concept of multi agent, several questions should be answered. From aspects regarding to cognitive features (Artificial Intelligence techniques used to provide the smart side of agents) to the details of communication protocol and agreements among components need to be carefully defined and specified.
The way how these features is implemented or specified depends, among other things, on the complexity of the agent, the available resources, and the preferences of the designer. Simple agents, for example, can be implemented equally efficiently in competitive and cooperative systems. However, as more complex agents are created and issues such as attentional mechanisms and modelling of mental processes are explored, a single arbitration mechanism may not be suitable [Andronache, 2004]
Artificial intelligence (AI) efforts to design control systems for artificial agents are relying increasingly on research in the field of agent architectures. Various architecture schemes and design methodologies have been proposed, which focused on different aspects of agent control. Two main paradigms might be considered, behavior based architectures and cognitive based architectures. By nature, cognitive architecture are not targeted towards a particular agent or type of agent. However, that is not the case with behavior-based architectures, most of which were developed for particular kinds of agents or targeted at a particular class of tasks. Both of the paradigms have examples of architectures which have been successful on their respective application domains. It would be advantageous if components and principles which contributed to their success could be reused in other circumstances. An especially interesting question is whether these principles and components could be utilized in other architectures that do not use the same basic components or design methodology.
A first problem related to determine whether an architecture is successful or not is credit assignment: it may be difficult to say what part of an architecture or design accounts for its success. A second arises from the difficulty of comparing two different architecture types directly, because their design assumptions and domain restrictions may vary significantly, e.g., symbolic versus «sub» or non-symbolic, high-level versus low-level, serial vs. parallel, software versus robotic agents. A third

problem is that the characteristics which impede direct comparison also hinder the combination of mechanisms into a unitary architecture. Consequently, it is difficult if not impossible not only to assess the advantages and disadvantages of particular designs and methodologies but also to utilize them in other designs without a common language or framework in which architectures could be compared.
Another problem, specific to current behavior-based architectures, is that the mechanisms used for behavior selection are typically fixed. While it may be possible to adjust some of the mechanisms' parameters to make them more adaptive, they cannot be changed altogether.
As a step towards the development of a general framework for agent architectures, Andronache [Andronache, 2004] introduces the APOC architecture framework. APOC is not only intended as a theoretical framework, which allows researchers to analyze, evaluate, and compare agent architectures, but also, complemented by ADE, the APOC development environment, it functions as a practical tool for the design of complex agents.
In this thesis, for a formal and structured definition of the components for the proposed architecture, several componentes from APOC will be taken. In order, the follow aspects will be detailed discussed:

- Component notion.
- Link types.
- Associated processes.
- Activation function and priorities.

Figure **4-1** illustrates how an APOC component can be understood and specified.

4.2 Components architecture.

This section will introduce the architecture which will be proposed in this thesis. Figure **4-2** illustrates the actual components of the architecture in an abstract representation. The oval and squares represent devices containing the components of the architecture. The components are the following:

- **Knowledge Database (KDB):** It is a relational database storing the information collected by the monitoring agents (depicted on blue) and the criteria used to control the monitoring agents; These criteria would define conditions causing the agents react to perform (or not perform) some action, and would be modified according to the collected data.
- **Data Collector Agent (DCA):** It is depicted on red. It is the component on charge to receive information from the Monitoring Agents, and to send back control information, used to trigger specific actions to be performed by the monitoring agent, or to modify the conditions to observe on the resource which is being monitored by the agent.
- **Monitoring Agent (MA):** They are depicted on blue. These are agents residing at the system device to be monitored, and they are implemented to control and monitor a specific

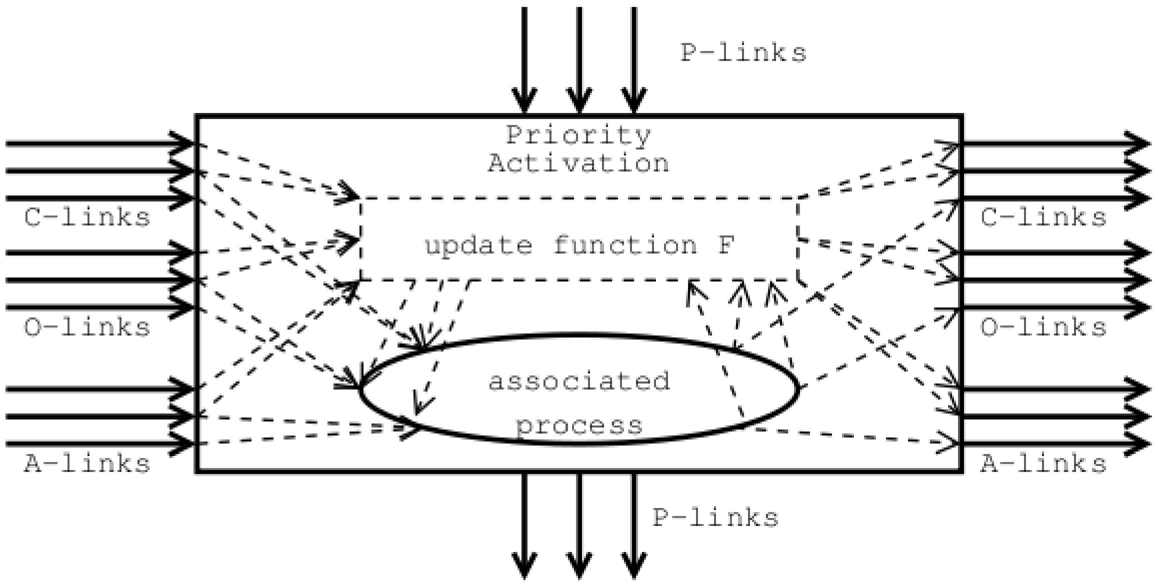

Figure **4-1**: An APOC component [Andronache, 2004]

resource on that system. The monitoring agent has the capacity to perform actions on that resource, and from the information collected during the monitoring process, is able to prevent failures or buggy behaviors. The criteria defining actions to perform, and behaviors to observe are defined and sent from the DCA. The agent is implemented in such a way that it is capable to be operative altough communication link with the DCA be lost. An important feature of the MA is its capacity to move itself onto a remote system device to monitor the specified resource.

- **Managed Resource (MR):** The managed resource is any entity within the system device under monitoring. That entity can be a hardware resource (Network Interface, Hard disk device, communications port), a logical resource (Filesystem, network connection) or a system process.

In the next section, the components of the proposed solution will be described in a more formal way, according to concepts defined in APOC reference architectural framework [Andronache, 2004].

4.3 Components in deep.

As mentioned on the first section of this chapter, the architectural framework defined to use on the construction of the proposed solution was APOC [Andronache, 2004]. APOC is an acronym standing «Activating, Proccessing, Observing, Components», which summarizes the four milestones of the framework. Tightly related to this milestones, appears the concept of communication links. APOC defines four types of links for the components, one for each one of the milestones. These four types are:

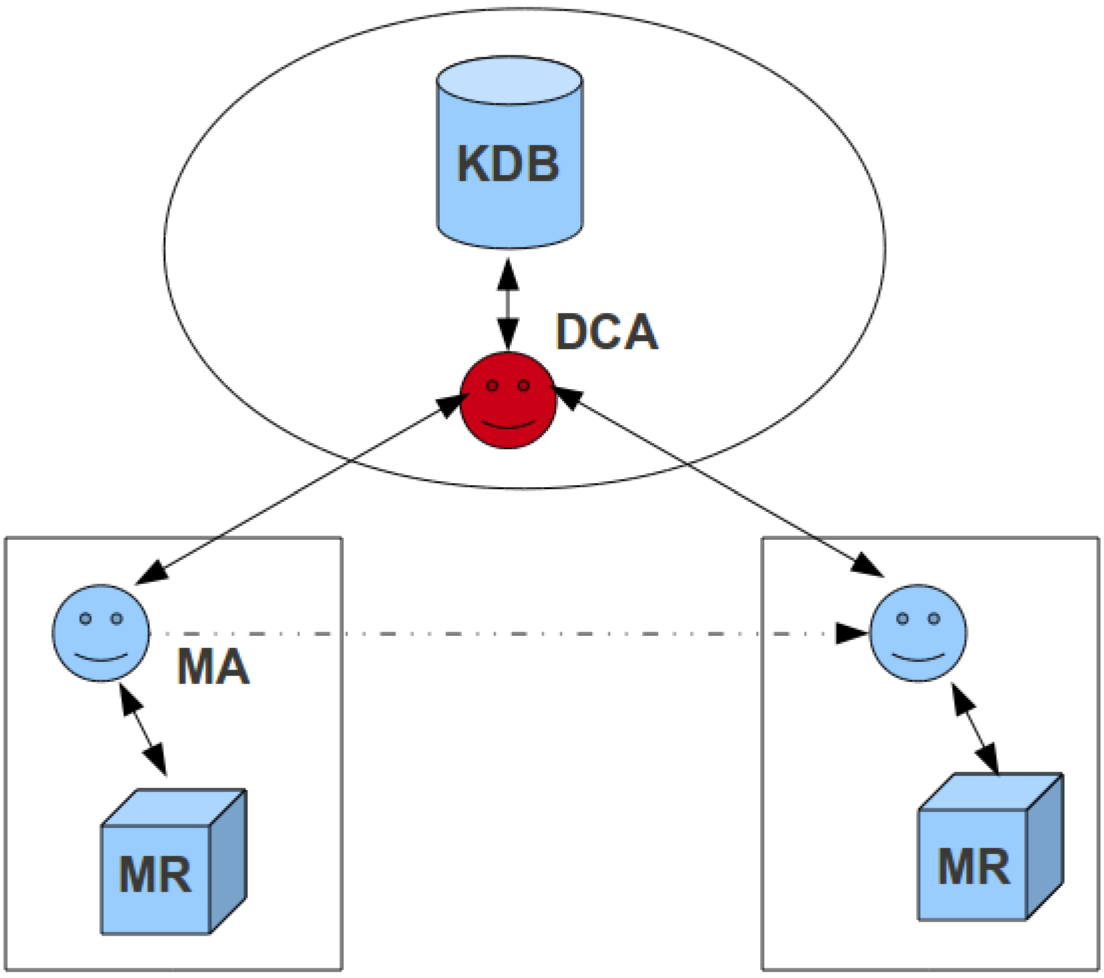

Figure **4-2**: Diagramatic architecture of the proposed solution.

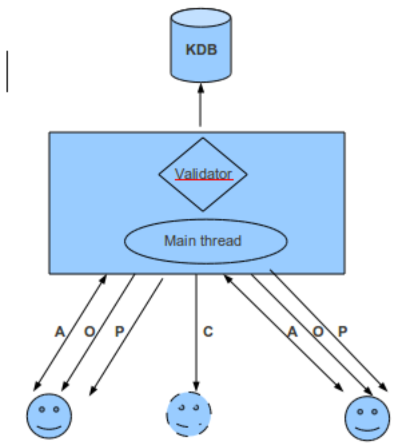

Figure **4-3**: Data Collector Agent

- Activation links (A-links): This kind of links allow to agents exchange general purpose messages.
- Process control links (P-liks): This type of links are used to influentiate the behavior of other components.
- Observation links (O-Links): Links which are used to observe the state of other components of the system.
- Component links (C-links): Links used to instatiate other components in the system, and then connect them by any of the other three types.

In the next subsections, the main two components of the architecture (DCA, and MA) will be described in detail according to APOC, and it will be introduced how these components will exhibit the eight principles of Autonomic Computing described by Horn [Horn, 2001].

4.3.1 Data Collector Agent

Figure **4-3**illustrates the diagramatic view of DCA agent.
Following APOC, the following elements can be identified.

- The **component** is the Data Collector Agent itself.
- As will be discussed in chapter 5, the environment that will be used in implementation is JADE, a framework for Multi Agent Systems implementation, based on JAVA programming language. Because of that choice, the **associated process** for this component is the main thread of the agent implementation.
- The DCA is the main agent in the system, and is the one in charge to control the other agents of the system. For that reason, it has no **activation function** and it has **highest priority** of all the agents**.**

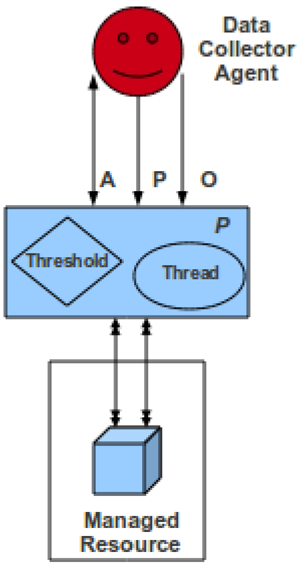

Figure **4-4**: Monitoring agent.

- **Activation** links correspond to the bidirectional interface of communication between DCA and MA's. Through this link, the messages with informational data coming from the monitoring agents are sent.
- **Observation** links are used for the DCA to query particular data and information state from MA.
- **Processing** links are used to modify the observation criteria on the MA, instruct the agents not to execute or execute selectively actions on the resource, or to request moving, cloning or shutdown of them. The DCA **component** does not receive directives through any **processing links** as it is the main component. The control of this component is under charge of the operating system of the device where it runs on.
- **Component** links are used for the instatiation of new MA.

4.3.2 Monitoring Agent

Figure **4-4** presents the diagram of Monitoring Agent (MA). Following APOC, the elements which can be identified on this component are:

- The **component** is the monitoring agent itself.
- As will be discussed in chapter 5, the environment that will be used in implementation is JADE, a framework for Multi Agent Systems implementation, based on JAVA programming language. Because of that choice, the **associated process** for this component is the main thread of the agent implementation.

- Every MA will have a **priority** indicator to be used by DCA, to resolve conflicts regarding what particular action to perform for handling or avoiding a failure detected on the Managed Resource (MR).
- Every MA will have defined a series of thresholds, used to specify when the MA has to **activate** a particular action for handling events or conditions on the MR. These thresolds have initial values when the MA is started or created, and they can be modified by the DCA, or by the MA itself.
- **Activation** links correspond to the bidirectional interface of communication between the DCA and the MA. Through these links data related to information collected or detected on the MR are exchanged.
- **Proccessing** links are used by the DCA to perform control operations on the MA. These operations include modification of the criteria used by the MA during operation, passivation of the MA, inhibition, shutdown, moving request or cloning request.
- **Observation** links are used by the DCA to query particular data and information state from MA.
- **Component** links do not apply for MA.
- MA interacts with the MR by means of particular interfaces provided by it. These interfaces can be system calls, files, management protocols, etc.

4.3.3 Exhibiting Autonomic Computing principles.

Following to Horn [Horn, 2001], Autonomic Computing is based on eight principles:

- ***Self-Awareness:*** This is, the system should know itself. It will need detailed knowledge of its components, current status, ultimate capacity, and all connections with other systems to govern itself. It will need to know the extent of its «owned» resources, those it can borrow or lend, and those that may be shared or should be isolated.
- ***Self-Configuration:*** System configuration or "setup" must occur automatically. Also, the system must modify itself, in such a way that its configuration be the most adequate to cope with environment conditions.
- ***Self-Optimization:*** This is, the system will always try to find other ways to improve its functioning. It will monitor its constituent parts and fine-tune workflow to achieve predetermined system goals.
- ***Self-Healing:*** System must be able to recover from events that might cause malfunctioning. The system also must be able to detect problem or potential problems, and according to this, define alternate ways to perform, applying reconfiguration to keep the system working.
- ***Self-Protection:*** Starting from the fact of the potentially agressive and hostile environment where system resides, it must be able to detect, protect and identify potential attacks or vulnerabilities, so that it can protect itself and keep working in a secure and consistent state.

- ***Self-adaptability:*** This is, the system must know its environment, the context which surrounds it when operating, and the other entities cohabitating with it. It must be able to adapt to this environment, and its changing conditions, by reconfiguring itself or optimizing itself.
- ***Openness:*** The components in system must be open to communicate each other, and must be able to work with shared technologies. Propietary solutions are not compatible with Autonomic Computing philosophy. Components should employ standard protocols, or at least *de facto* standards in their communication.
- ***Self-containment:*** Components within an Autonomic Computing System must be able to perform the task or tasks they have assigned, not requiring external interventions for the performing itself, and hiding the complexity to end user. This does not mean that the system will lack of interfaces allowing the intervention and feedback from human operators.

In the next two subsections, it will be clarified how the two main components of the proposed solution exhibit the principles of autonomic computing on their design.

4.3.4 Autonomic features of the Data Collector Agent:

- ***Self-Awareness:*** At the moment to be started, the DCA will become aware of its own existence, and it will have complete control of the MA associated.
- ***Self-Configuration:*** The DCA will have the capacity to modify parameters regarding the polling frequency of the MA, the criteria to trigger changes on the MA, and whether passivate them, induce additional actions, or shutdown them.
- ***Self-Optimization:*** The DCA will be able to modifiy its timing parameters, to not overwhelm the MA with very frequent queries of information or status, or with queries reporting high volume of data.
- ***Self-Healing:*** DCA will have the capacity to avoid error conditions caused by absent or invalid data, besides recovery procedures for situations such as database connection lost, or MA P-links lost (See section 4.3.1 about the meaning of P-Link term)
- ***Self-Protection:*** DCA will have validation mechanisms for any interaction coming from outworld, and for the data coming from MA, to avoid data corruption or invalidation.
- ***Self-adaptability:*** The adaptability of DCA will be represented by the Self-Optimization and the Self-Configuration. In situations such as slow answer from MA, DCA will be able to reduce the query rate of information from them.
- ***Openness:*** DCA will be implemented with an open and standard technology as JADE (See chapter 5) and using standard network protocols and interfaces.
- ***Self-containment:*** The task of DCA is clearly defined. It will be on charge of receiving and analyzing the information coming from MA, to produce criteria modifications on the actions

to be performed by them on the Managed Resources. Also, the DCA will be on charge to track the activities performed on the MA.

4.3.5 Autonomic features of the Monitoring Agent:

- ***Self-Awareness:*** At the moment to be started, the MA will become aware of its own existence, and will perform a kind of registration on the DCA, to establish the control relationship. Also, the MA will have to establish a relationship with the MR which will be monitoring and controlling.
- ***Self-Configuration:*** The MA will be able to modify the criteria to detect failures or buggy behaviors with the goal to anticipate and prevent them. These eventual modifications will be notified to the DCA, so that they be stored on the Knowledge Database. Also, the MA will have the capacity to perform even when communication links with the DCA be lost.
- ***Self-Optimization:*** MA will modify the frequency and parameters of the procedures it performs to monitor the MR, so that it prevents to become intrusive on the MR itself.
- ***Self-Healing:*** MA will have the capacity to avoid error conditions caused by absent or invalid data coming from the MR, besides recovery procedures for situations such as MR connection lost, or communication links lost (See section 4.3.2 about the meaning of P-Link term)
- ***Self-Protection:*** MA will have validation mechanisms for any interaction coming from out-world, and for the data coming from DCA, to avoid data corruption, invalidation or intrusive actions on the MR. MA will have to certify it will only receive information coming from the DCA of the system.
- ***Self-adaptability:*** The adaptability of MA will be represented by the Self-Optimization and the Self-Configuration. In situations such as slow answer from MR, MA is capable to reduce the query rate of information to avoid intrusive behavior.

4.4 Detailed description of platform running

This section will provide a description about how the platform runs, and performs the monitoring and fault management tasks.

4.4.1 Platform Initialization: DCA startup.

The first component to be started is the DCA, which operates as the master component (It is assumed that the KDB is available when DCA is going to be started). It has to be availabile for the initialization of monitoring agents, although it is not required for their later running. When DCA is started, it is ready to send and receive information from/to the monitoring agents. DCA reads database information, and gets ready to offer initialization parameters to the agents which perform successful authentication against it.

4.4.2 Monitoring agents initialization.

After DCA is running, the remote monitoring agents may be initiated. The first task executed by monitoring agents after being started is the registration against DCA. The registration is executed after the monitoring agent gets authenticated against the DCA. If registering is successful, DCA confirms the registration by returning to the monitoring agent the parameters so that it starts to execute its task. Also, if available, DCA can return criteria to be used by the monitoring agent to detect failures or to take actions when failure occurs. In case there not be information regarding actions to be taken when failures occur, the default behavior of the agents is just notify back the failure condition to DCA.

4.4.3 Monitoring and failure detection.

The agents execute a continous loop performing the monitoring action, according to agent implementation. This action, guided by the criteria which might have been pased at initialization time, allows to detect whether the monitored resource is in correct status, or whether a failure condition occurs. If a fault is detected, the default behavior of the agent, when there is not any action defined for it, is notify the DCA. DCA might return to monitoring agent an action specification informing it what to do to try to solve the fault. Also, DCA might ask for further information to monitoring agent, to take a better decision regarding the action to ask tio be performed by it.
The monitoring agent, when receives an action notification from DCA, executes it, and verifies whether the fault condition was solved. If solved, monitoring returns to initial state. If not solved, the monitoring agent notifies again to DCA, asking for additional actions to perform.
When DCA notifies actions to be performed, the monitoring agent stores it into a local action memory. This memory allows to agent to know what actions to perform when fault conditions occur, without having to ask them to DCA. Monitoring agent would apply the actions stored into that memory, before having to ask to DCA.

4.4.4 Remote monitoring at additional locations.

After correcting a fault, the monitoring agents, if instructed, may move (or clone) onto another locations (other monitored resources), and perform the monitoring tasks. If fault is detected, the monitoring agents may apply the actions they have stored into their local action memory, or they might ask to DCA for further actions to perform..

5 Experiments and results

5.1 Introduction

The present chapter describes de tests performed to validate the advantages of the proposed approach for fault management. For these tests, a preliminary implementation of the proposed approach was performed, and run on a typical platform for the voice over IP Service. Two aspects were assesed. The time required to detect a failure, and the bandwith consumed.

5.2 Description of test platform

For the validation of the approach proposed in this work, a typical platform in the Voice over IP Services was chosen. This kind of platform was chosen, because of its criticity, and the relationships established between the components, which tend to induce failures in the other components, when a problem occurs.
The figure **5-1** depicts the testbed which was used for the implementation.

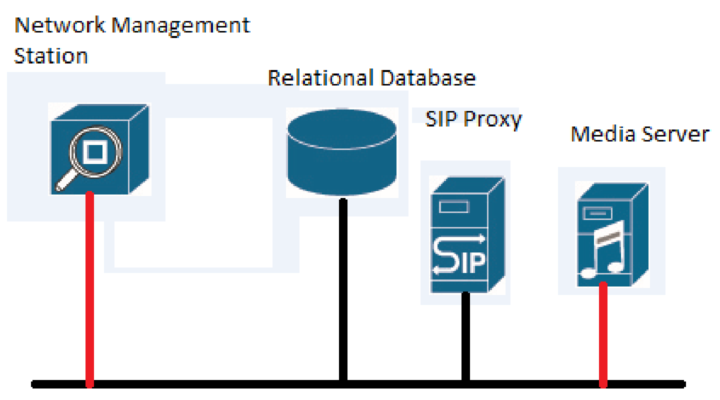

Figure **5-1**: Testbed used to validate and test the proposed approach.

All of the machines run Linux Operating System, with no special settings. Network Management Station has installed the NET SNMP software, for the tests based on SNMP, MySQL for the database used by the Data Collector Agent of the proposed approach, and JAVA development kit for the Agent Platform. Relational Database is MySQL, with standard configuration, and contains the service database which is used by the SIP Proxy and the Media Server. SIP Proxy is based in OpenSIPS software, an open implementation of SIP Protocol, and Media Server is based on Asterisk, a full SIP application server which provides different telephony servers. Besides the specific purpose software deployed on each node, JAVA Development Kit for running JADE is installed, and NET-SNMP Agent, for the SNMP Based tests.

5.3 Agents implementation.

For the implementation of the agents, the used tool was JADE [Jad,]. JADE is a middleware that facilitates the development of multi-agent systems. It includes:

- A runtime environment where JADE agents can «live» and that must be active on a given host before one or more agents can be executed on that host.
- A library of classes that programmers have to/can use (directly or by specializing them) to develop their agents.
- A suite of graphical tools that allows administrating and monitoring the activity of running agents.

A very important concept in JADE terminology is *container.* Each running instance of the JADE runtime environment is called a Container as it can contain several agents. The set of active containers is called a Platform. A single special Main container must always be active in a platform and all other containers register with it as soon as they start. It follows that the first container to start in a platform must be a main container while all other containers must be «normal» (i.e. non-main) containers and must «be told» where to find (host and port) their main container (i.e. the main container to register with).
JADE provides several API's for the agent implementation, and for details such as their behaviors and their communications mechanisms.
For accessing the resources and the detailed status information of monitored resources, the Java Native Interface (JNI) was used. JNI is an interface of Java programming language, which allows to execute methods written in native language of the platfrom where Java virtual machine runs on. In this implementation, the native code was written in ANSI C language, to invoke the Linux Operating System syscalls required for the monitoring tasks performed by the agents.
The election of JNI as the mechanism to interact with the monitoring resource information is not gratuitous. During the development of this work, several tests were performed to assess different schemas to access system informations. The work by Branch and Gutierrez [Branch and Gutierrez, 2011] presents a comparison where, although Python programming language offers the best behavior, because of the election of JADE as the framework to implement the agents, JNI is more suitable to integrate within it.

5.4 Experiments and results

For the testing, three scenarios where defined and tested. The three scenarios consist in inducing failures on a Resource node, and measuring the time used to detect (and correct) the failure.

5.4.1 Full filesystem

On the first scenario, the first failure to be induced was a filesystem full on Relational Database. The failure was induced by creating an increasing size file, by means of an infinite loop script which concatenated 1MB files to an existing file until the whole filesystem was filled.
The SNMP based detection was performed by running a shell script which invocated *snmpwalk* system command once every second, querying the OID referencing the filesystem to be monitored.
For the agent scenario, platform was started up, and a monitor agent was created on the supervised host, indicating it what filesystem to monitor. For this first test, no action was defined for the monitoring agent at startup time. At failure ocurrence, DCA instructed the agent the action to perform, which in in this case was deleting the file causing the filesystem full.
Table **5-1** presents the results of the execution of this test scenario.

Variable	**SNMP TEST**	**AGENT TEST**
Elapsed time	3 mins, 42 sec	4 mins, 4 sec
Total packets	811	318
Average Bandwidth (Mbps/s)	0.003	0.002
Average packet size (Bytes)	87.89	155.08
Average packets per second	3.642	1.303

Table **5-1**: Data of the test of full filesystem fault at database.

5.4.2 Dead system process.

The second test scenario is the detection of a dead process within the SIP Proxy. The failure is induced by killing the process, disappearing it from system processes table.
The SNMP based detection was performed by running a shell script which invocated *snmpwalk* system command once every second, querying the OID referencing the entry of the process within the system process table exposed through SNMP.
For the agent scenario, platform was started up, and a monitor agent was created on the supervised host, indicating the name of the executable associated to process. In this case, a single process was montitored, although the approach is valid for multiple instances of same process. For this test, no action was defined for the monitoring agent at start time. At failure ocurrence, DCA instructed the monitoring agent to restart the process.
Table **5-2** presents the results of the execution of this test scenario.

Variable	**SNMP TEST**	**AGENT TEST**
Elapsed time	1 min	1 min
Total packets	508	147
Average Bandwidth (Mbps/s)	0.003	0.002
Average packet size (Bytes)	87.89	155.08
Average packets per second	3.642	1.303

Table **5-2**: Data of the test of dead system process at SIP proxy.

5.4.3 Network interface missconfiguration.

The second test scenario is the detection of a missconfiguration within the Mediaserver. The failure is induced by modifying the network interface configuration, changing the duplex mode from full to half. This is a very common error on this kind of equipments, and usually causes malfunction in the service it offers.
The SNMP based detection was performed by running a shell script which invocated *snmpwalk* system command once every second, querying the OID referencing the duplex mode of the main network interface of the mediaserver. In this case, the monitoring script was capable, after detecting the error, to perform a correction by setting the parameter of duplex mode, by executing *snmpset* system command.
For the agent scenario, platform was started up, and a monitor agent was created on the supervised host, indicating the name of the main network interface in the host. For this test, no action was defined for the monitoring agent at start time. At failure ocurrence, DCA instructed the monitoring agent to reconfigure the network interface. Table **5-3** presents the results of the execution of this test scenario.

Variable	**SNMP TEST**	**AGENT TEST**
Elapsed time	2 min	2 min, 32 secs
Total packets	203	84
Average Bandwidth (Mbps/s)	82.6	38.7
Average packet size (Bytes)	87.89	155.08
Average packets per second	3.642	1.303

Table **5-3**: Data of the test of network interface missconfiguration at mediaserver.

5.5 Experiments discussion.

In this section, a discussion on the results obtained from the tests will be presented.
A very important point to state is the fact that in the first two tests, besides of the fault detection, from the information that DCA returns at failure occurrence, the agent is capable to perform an action to correct the fault condition.
The concept behind the implemented platform is providing the agents with logic and actions (behaviors) allowing that, from the information provided by the DCA in this case, they be capable to apply corrections.
In the standard implementation, SNMP based detection is not capable to correct complex faults as it should have external mechanisms to apply any correction. SNMP can solve very specific faults, if the correction consists in modifying a single variable, as seen on the third scenario, where the script could restore the network interface configuration. In the other cases, an autonomous correction y means of SNMP is not possible, and the intervention of a human operator is required. The fault correction is reactive, not proactive.
The agents implementation, by default is capable to provide mechanisms to apply corrections when particular types of faults are detected. Also, because of the local memory the agents have, they

can try to apply corrections when faults are detected, without the need to ask for information to DCA, in an autonomous way. The agents, provided information to react to fautls are capable to apply corrections in a proactive way. The tradittional approaches, based on SNMP, are just based on notificacions, and focused on alarm and notification triggering, leaving the action performed to correct the fault condition itself to an external entity (usually a human operator).
It can be observed that the elapsed time, measured since monitoring was started until the fault condition was detected (and corrected), is a longer for agents than for SNMP. This can be explained as the agent platform (main container and satellital container) require some time for its initialization. In SNMP case, monitoring starts by just run the script executing *snmpwalk*.
The average packet size is also greater for agents than SNMP because the network transport protocol that agent platform uses. JADE uses TCP for the communication, different to SNMP, which uses UDP. The usage of TCP as communication protocol in the agent platform increases the reliability of monitoring, as delivery of messages is guarateed, and monitorin agents do not require to have additional logic for handling message delivery.
Another fact which can be observed is that agent approach require less packets to be exchanged. As the agent communicates with DCA only for notifications and initialization tasks, and not for monitoring *per se*, less traffic is required. For SNMP, more packtes are required, as the decision to detect the fault is taken at monitoring station, not in the monitored resource itself.
Briefing, from the point of view of the two variables assesed, the proposed approach, based on agents, shows to be more efficient, by exhibiting lower times to detect the fault, less traffic requirements, and besides, provides added value by the capability of executing and performing proactive corrective actions, to eliminate and avoid the fault conditions.

6 Conclusions and future work

6.1 Conclusions

In this work, an approach for fault management using Autonomic Computing and Mobile Agents has been proposed. This approach has showed to offer more advantages than conventional approaches by exhibiting the following features.

- From the architectural point of view, it is not server centric, as traditional systems, usually based on a master component which queries sattelital agents to obtain information. Although the proposed approach has a master component, the monitoring agents do not require the communication with this component to perform their tasks, as they are designed to operate in autonomous way.
- It provides self-learning capacities. Traditional systems just report the fault by triggering an alarm when a threshold is reached, or when the fault itself arises, but they lack of efficient mechanisms to correct the fault. The agents in the proposed approach have the capacity to analyze causes of the detected problems, and perform further actions to avoid the fault occurence.
- It provides a richer communication protocol among the involved entities, allowing to have a detailed information regarding facts and actions to be performed. The concept of agents ontology provides a very robust mechanism for the communication protocol regarding message formats and interpretation. Traditional approaches are based on a simple request/response protocol, with very strict and limited message formats.
- It provides mobility, which allows to monitoring agents to move accross the network to detect and correct faults. Traditional systems are based on static monitoring systems, which are confined to run at a single host.

These features present a system which is very suitable for platforms as the ones used in Telecommunications service, which nowadays require high levels of availability because of the criticity of the services they offer. Also, the implemetation of the features previously mentioned contribute to improve the task of fault management, by providing added value to monitoring tasks, and correction for some of the limitations of traditional approaches.

6.2 Future work

There are several points to have present, which could be starting point for future work:

- Include root cause analysis by using for example Belief Bayesian Networks on the agents, to offer sophisticated diagnosis features.
- Reduce the memory footprint of the agents, to make them suitable on systems and platforms with reduced hardware resources.
- Provide more robust authentication and authorization mechanisms for the communication between monitoring agents and the KDB.
- Provide more secure communication protocols among the platform components.
- Include patter recognition techniques to perform sophisticated analysis on the data collected and stored into KDB.
- Define additional metrics which allow to compare and expose the goodness of multi agent and autonomic approach in comparison to traditional and centric fault management solutions.

Bibliography

[TMF,] Enhanced telecom operations map (etom), the business frame work, release 7.0.

[ISO, a] Information processing systems - open systems interconnect - basic reference model: Part 4 - management framework, 1989.

[ISO, b] Information technology - service management - part 1: Specification.

[ITI,] Infrastructure library ITIL.

[Jad,] *JADE Tutorial: JADE Programming for Beginners.*

[Adhicandra et al., 2003] Adhicandra, I., Pattinson, C., and Shaghouei, E. (2003). Using mobile agents to improve performance of network management operations.

[Aghasaryan et al., 1997] Aghasaryan, A., Fabre, E., Benveniste, A., Boubour, R., and Jard, C. (1997). A petri net approach to fault detection and diagnosis in distributed systems,. In *Proc. 36th IEEE Conference on Decision and Control, IEEE (CDC 97).*

[Al-Fuqaha et al., 2009] Al-Fuqaha, A., Rayes, A., Guizani, M., Khanvilkar, M., and Ahmed., M. (2009). Intelligent service monitoring and support. In *Proc. IEEE International Conference on Communications, 2009. ICC '09.*

[Al-Kasassbeh and Adda, 2008] Al-Kasassbeh, M. and Adda, M. (2008). Analysis of mobile agents in network fault management. *Journal of Network and Computer Applications*, 31:699–711.

[Andronache, 2004] Andronache, V. (2004). *APOC AND ADE: THEORY AND PRACTICE IN THE DESIGN OF ARCHITECTURES FOR BEHAVIOR-BASED AGENTS.* PhD thesis, University of Notre Dame.

[Baldi and Picco, 1998] Baldi, M. and Picco, G. (1998). Evaluating the tradeoffs of mobile code design paradigms in network management applications. In *Proc. of 20th Internation Conference on Software Engineering.*

[Bhattacharyya et al., 2004] Bhattacharyya, S., Huang, Z., Chandra, V., and Kumar, R. (2004). A discrete event systems approach to network fault management: detection & diagnosis of faults. In *Proceedings of the 2004 American Control Conference.*

[Boudriga and Obaidat, 2004] Boudriga, N. and Obaidat, M. (2004). Intelligent agents on the web: a review. *Computing in Science and Engineering*, 06:35–42.

[Branch and Gutierrez, 2011] Branch, J. and Gutierrez, S. (2011). A comparison of performance to access programatically system resources in linux. *Revista Avances en Sistemas e Informatica, Vol.8 No.1, marzo de 2011 - Medellin.*, 8:39–42.

[Brenner, 2006] Brenner, M. (2006). Classifying ITIL processes: A taxonomy under tool support aspects. In *First IEEE/IFIP International Workshop on Business-Driven IT Management (BDIM 2006).*

[Brodie et al., 2002] Brodie, M., Rish, I., and S.Ma (2002). Intelligent probing: a cost-effective approach to fault diagnosis in computer networks. *IBM Systems Journal*, 41:372–385.

[Callaway et al., 2008] Callaway, R., Devetsikiotis, M., Viniotis, Y., and Rodriguez, A. (2008). An autonomic service delivery platform for service-oriented network environments. In *IEEE International Conference on Communications, 2008. ICC '08.*

[Chen, 2005] Chen, Z. (2005). Proactive probing and probing on demand in service fault localization. *The international journal of Intelligence control and systems.*, 2:107–113.

[Chen, 2006] Chen, Z. (2006). Service fault localization using probing technology. In *Proceedings of the 2006 IEEE International Conference on Networking, Sensing and Control, 2006. ICNSC '06.*

[Chu et al., 2009] Chu, L. W., Zou, S. H., Cheng, S. D., Wang, W. D., and Tian, C. Q. (2009). Internet service fault management using active probing in uncertain and noisy environment. In *Proc. Fourth International Conference on Communications and Networking in China, ChinaCOM 2009.*

[Franklin and Graesser, 1996] Franklin, S. and Graesser, A. (1996). It's an agent or just a program?: A taxonomy for autonomous agents. In *Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages,.*

[Fujisaki et al., 1997] Fujisaki, T., Hamada, M., and Kageyama, K. (1997). A scalable fault-tolerant network management system built using distributed object technology. In *Proc. First International Enterprise Distributed Object Computing Workshop.*

[Gavalas et al., 2009] Gavalas, D., Tsekouras, G. E., and Anagnostopoulos, C. (2009). A mobile agent platform for distributed network and systems management. *The Journal of Systems and Software*, 82:355–371.

[Gorod et al., 2007] Gorod, A., Gove, R., Sauser, B., and Boardman, J. (2007). System of systems management: A network management approach. In *IEEE International Conference on System of Systems Engineering.*

[Goyal et al., 2009] Goyal, P., Mikkilineni, R., and Ganti, M. (2009). Fcaps in the business services fabric model. In *18th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises.*

[Harrington et al., 2002] Harrington, D., Presuhn, R., and Wijnen, B. (2002). Rfc 3411: An architecture for describing simple network management protocol (snmp) management frameworks. Technical report, IETF.

[Harrison et al., 1995] Harrison, C., Chess, D., and Kershenbaum, A. (1995). Mobile agents: Are they a good idea? Technical report, IBM T.J. Watson Research Center.

[Horn, 2001] Horn, P. (2001). Autonomic computing: IBM's perspective on the state of information technology. Technical report, IBM Corp.

[Hwang et al., 2010] Hwang, I., Kim, S., Kim, Y., and Seah, C. E. (2010). A survey of fault detection, isolation, and reconfiguration methods. *IEEE Transactions on Control Systems Technology*, 18:636–653.

[IBM Corp., 2005] IBM Corp. (2005). An architectural blueprint for autonomic computing. Technical report, IBM Corp.

[Ionescu et al., 2008] Ionescu, D., Solomon, B., Litoiu, M., and Mihaescu, M. (2008). A robust autonomic computing architecture for server virtualization. In *International Conference on Intelligent Engineering Systems, 2008. INES 2008.*

[ITSMF, 2003] ITSMF (2003). *IT Service Management: An Introduction.* Van Haren Publishing; 3Rev Ed edition (March 13, 2003).

[Jennings and Wooldridge, 1998] Jennings, N. and Wooldridge, M. (1998). *Applications of Agent Technology in Agent Technology.* Springer-Verlag.

[Jin and Liu, 2004] Jin, X. L. and Liu, J. M. (2004). *Agents and Computational Autonomy: Potential, Risks, and Solutions.*, chapter From Individual Based Modeling to Autonomy Oriented Computation, pages 151–169. University of Bath.

[Kerschberg et al., 1991] Kerschberg, L., Baum, R., Waisanen, A., Huang, I., and Yoon., J. (1991). Managing faults in telecommunications networks: A taxonomy to knowledge-based approaches. In *Proc. IEEE International Conference on Systems, Man, and Cybernetics, 1991. 'Decision Aiding for Complex Systems'.*

[Kirmani and Hood, 2004] Kirmani, E. and Hood, C. (2004). Diagnosing network states through intelligent probing. In *Network Operations and Management Symposium, 2004. NOMS 2004. IEEE/IFIP.*

[Labib, 2006] Labib, A. W. (2006). Next generation maintenance systems: Towards the design of a self-maintenance machine. In *IEEE International Conference on Industrial Informatics, 2006.*

[Li et al., 2009a] Li, C., Zou, S., and Lingwei, C. (2009a). Online learning based internet service fault diagnosis using active probing. In *Proc. IEEE International Conference on Networking, Sensing and Control, Okayama, Japan.*

[Li and Wu, 2009] Li, H. and Wu, Z. (2009). Research on distributed architecture based on soa. In *International Conference on Communication Software and Networks, 2009. ICCSN '09.*

[Li et al., 2009b] Li, Y., Zhao, C., and Yin, Y. (2009b). A method of building the fault propogation model of distributed application systems based on bayesian network. In *Second International Workshop on Computer Science and Engineering.*

[Lin et al., 2005] Lin, P., MacArthur, A., and Leaney, J. (2005). Defining autonomic computing: A software engineering perspective. In *Proc. of the 2005 Australian Software Engineering Conference (ASWEC 05).*

[Liu et al., 2001] Liu, J., Tsui, K., and Wu, J. (2001). Introducing autonomy oriented computation. In *Proceedings of First International Workshop on Autonomy Oriented Computation.*

[Magedanz, 2004] Magedanz, T. (2004). Report on fet consultation meeting on communication paradigms for 2020. Technical report, Fraunhoffer FOKUS.

[Mei-hui et al., 2010] Mei-hui, W., Chuan, L., Zhao, M., and Dong, Z. (2010). A discussion of using self-maintenance technology to achieve high reliability of equipment. In *Prognostics and Health Management Conference, 2010. PHM '10.*

[Miller and Arisha, 2001] Miller, R. and Arisha, K. (2001). On fault management using passive testing for mobile ipv6 networks. In *IEEE Global Telecommunications Conference, 2001. GLOBECOM '01.*

[Picco, 2001] Picco, G. P. (2001). Mobile agents: an introduction. *Microprocessors and Microsystems*, 25:65–74.

[Pras et al., 2004] Pras, A., Drevers, T., van de Meent, R., and Quartel, D. (2004). Comparing the performance of snmp and web services-based management. *IEEE Transactions on Network and Service Management*, 1:72–82.

[R.Sterrit, 2001] R.Sterrit (2001). Soft computing and fault management. In *Proceedings of Fourth International Symposium Soft Computing and Intelligent Systems for Industry, Paisley, Scotland, United Kingdom.*

[Salle and Bartolini, 2004] Salle, M. and Bartolini, C. (2004). Business driven prioritization of service incidents. In *Distributed Systems Operations and Management.*

[Samaras et al., 1999] Samaras, G., Pitoura, E., and Papastavrou, S. (1999). Mobile agents for www distributed database access. In *Proceedings of 15th International Conference on Data Engineering.*

[Sirvunnabood and Labib, 2005] Sirvunnabood, S. and Labib, A. (2005). Towards the development of self-maintenance machines. In *Proc. of the 2005 Third European Conference on Intelligent Management Systems in Operations. 28th and 29th June. Greater Manchester U.K.*

[Steinder and Sethi, 2004] Steinder, M. and Sethi, A. S. (2004). A survey of fault localization techniques in computer networks. *Science of Computer Programming*, 53.

[Sterritt, 2001] Sterritt, R. (2001). Discovering rules for fault management. In *Proc. Eighth Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems, 2001. ECBS 2001.*

[Sterritt and Bustard, 2002] Sterritt, R. and Bustard, D. (2002). Fusing hard and soft computing for fault management in telecommunications systems. In *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews,*.

[Sterritt et al., 2003] Sterritt, R., Bustard, D., and McCrea, A. (2003). Autonomic computing correlation for fault management system evolution. In *IEEE international conference industrial informatics*.

[Sterritt et al., 2002] Sterritt, R., Curran, E., and Song, H. (2002). Hacker: human and computer knowledge discovered event rules for telecommunications fault management. In *IEEE International Conference on Systems, Man and Cybernetics, 2002*.

[Sun et al., 2003] Sun, H., Han, J. J., and Levendel, H. (2003). Availability requirement for a fault-management server in high-availability communication systems. *IEEE transactions on reliability*, 52:238–244.

[Sun Microsystems INC, 2010] Sun Microsystems INC (2010). Predictive self healing. Technical report, Sun Microsystems INC.

[Tizghadam and Leon-Garcia, 2010] Tizghadam, A. and Leon-Garcia, A. (2010). Autonomic traffic engineering for network robustness. *IEEE journal on selected areas in communications*, 28:39–50.

[Tosic and Agha, 2004] Tosic, P. T. and Agha, G. A. (2004). Towards a hierarchical taxonomy of autonomous agents. In *IEEE International Conference on Systems, Man and Cybernetics*.

[Umeda et al., 1995] Umeda, Y., Tomiyama, Y., and Yoshikawa, H. (1995). A design methodology for self-maintenance machines. *ASME Journal of Mechanical Design*, 177.

[Uthurusamy, 1996] Uthurusamy, R. (1996). From data mining to knowledge discovery: Current challenges and future directions. *Advances in knowledge discovery and Data Mining*, pages 561–569.

[V. Baggiolini, 1998] V. Baggiolini, J. H. (1998). Generic fault management techniques. Technical report, University of Geneva, Switzerland.

[Varga and Moldovan, 2007] Varga, P. and Moldovan, I. (2007). Integration of service-level monitoring with fault management for end-to-end multi-provider ethernet services. *IEEE Transacions on Network and Service Management*, 4.

[Wang and Zhou, 2004] Wang, J. and Zhou, M. (2004). Providing network monitoring service for grid computing. In *Proc. 10th IEEE International Workshop on Future Trends of Distributed Computing Systems, 2004. FTDCS 2004.*

[Wong, 2009] Wong, D. (2009). Adapting e-TOM model to improve multimedia network management: A case study with a taiwan multimedia service provider. In *Fifth International Conference on Intelligent Information Hiding and Multimedia Signal Processing*.

[Wooldridge and Jennings, 1995] Wooldridge, M. and Jennings, N. (1995). *Intelligent Agents: Theory and Practice*. Knowledge Engin Rev.

[Yang et al., 1998] Yang, J., Pai, P., Honavar, V., and Miller, L. (1998). Mobile intelligent agents for document classification and retrieval: A machine learning approach. In *14th European Meeting on Cybernetics and Systems Research. Symposium on Agent Theory to Agent Implementation.*

[Yougang, 2009] Yougang, Z. (2009). Fault diagnosis model based on rough set theory and expert system. In *International Colloquium on Computing, Communication, Control, and Management.*

[Yubao and Renyuan, 2009] Yubao, M. and Renyuan, D. (2009). Mobile agent technology and its application in distributed data mining. In *First International Workshop on Database Technology and Applications.*

[Yuniarto and Labib, 2006] Yuniarto, M. N. and Labib, A. W. (2006). Fuzzy adaptive preventive maintenance in a manufacturing control system: a step towards self-maintenance. *International Journal of Production Research*, 44.

[Zaharia et al., 2003] Zaharia, M. H., Leon, F., and Galea, D. (2003). A framework for distributed computing using mobile intelligent agents. *New Trends in Computer Science and Engineering*, pages 219–223.

[Zaw and Soe, 2008] Zaw, M. P. P. and Soe, S. M. M. (2008). Design and implementation of client server network management system for ethernet lan. *World Academy of Science, Engineering and Technology*, 48.