# ANA Project

Autonomic Network Architecture


Sixth Framework Programme
Priority FP6-2004-IST-4
Situated and Autonomic Communications (SAC)

**Project Number: FP6-IST-27489**

Deliverable D.3.5v1

# Specification of the Failure-Detection and Fault-Management part of the ANA Architecture (v1)

**First specification of principles and mechanisms for Failure-Detection and Fault-Management**

Version 1

# ANA Project

## Autonomic Network Architecture


| | |
|---|---|
| **Project Number** | FP6-IST-27489 |
| **Project Name** | ANA - Autonomic Network Architecture |
| **Document Number** | FP6-IST-27489/WP3/D.3.5v1 |
| **Document Title** | Specification of the Failure-Detection and Fault-Management part of the ANA Architecture (v1) |
| **Workpackage** | WP3 |
| **Editor** | Ranganai Chaparadza (Fraunhofer FOKUS) |
| **Authors** | Ranganai Chaparadza (FOKUS) |
| **Reviewers** | Guy Leduc |
| **Dissemination level** | Public |
| **Contractual delivery date** | 31st December 2007 |
| **Delivery Date** | 31st December 2007 |
| **Version** | 1.0 |

**Abstract:**

This Deliverable presents our research work on the design of a Unified Framework for Implementing Autonomic Fault-Management and Failure-Detection for Self-Managing Networks that can be applied to ANA or similar projects related to autonomic networking.

The designed generic unified implementation framework addresses the following issues:

1. Definition of the components required in implementing Autonomic Fault-Management and Failure-Detection, as well as their roles.
2. Assignment of functions related to automated, co-operative and/or collaborative fault-management & failure detection, to the defined components.
3. Design of a generic architecture for the Unified Implementation Framework.
4. Capturing of some fundamental design and operational requirements imposed on normal functional blocks of a node by autonomic fault-management and failure detection, in order to enforce co-operation in processes such as the sharing of incident information (faults, errors, failures) among node-local, as well as distributed entities in the network.

In this Deliverable we also present in brief the inter-working between components for Fault-management and Failure-detection, and the Resilience and Monitoring Frameworks of ANA. This is shown on the architecture meant for the ANA node, which is derived from the generic architecture of the Unified Implementation Framework to be published by the Journal.

**Keywords:**

A **Unified Framework for Implementing Autonomic Fault-Management and Failure-Detection for Self-Managing Networks**, Components and Functional Requirements for the Unified Framework.

# Executive Summary

Today's network management as known within the FCAPS management framework [1], is moving towards the definition and implementation of "self-managing" network functions, with the aim of eliminating or drastically reducing human intervention in some of the complex aspects or daunting tasks of network management. The Fault-management plane of the FCAPS framework [1] deals with the following functions: *fault-detection, fault-diagnosis, localization or isolation, and fault-removal*. Task automation is at the very heart of self-managing (autonomic) nodes and networks, meaning that all functions and processes related to fault-management must be automated as much as possible within the functionalities of self-managing (autonomic) nodes and networks, for us to talk about autonomic fault-management. At this point in time there are projects calling for implementing new network architectures that are flexible to support on-demand functional composition for context or situation aware networking. A number of such projects have started, under the umbrella of the so-called clean-slate network designs. Therefore, this calls for open frameworks for implementing self-managing (autonomic) functions across each of the traditional FCAPS management planes [1]. This Deliverable reports on our work on designing a **unified framework for implementing autonomic fault-management and failure-detection for self-managing networks [35]**.

The designed generic unified implementation framework addresses the following issues:

1. Definition of the components required in implementing Autonomic Fault-Management and Failure-Detection, as well as their roles.
2. Assignment of functions related to automated, co-operative and/or collaborative fault-management & failure detection, to the defined components.
3. Design of a generic architecture for the Unified Implementation Framework.
4. Capturing of some fundamental design and operational requirements imposed on normal functional blocks of a node by autonomic fault-management and failure detection, in order to enforce co-operation in processes such as the sharing of incident information (faults, errors, failures) among node-local, as well as distributed entities in the network.

In this Deliverable we also present in brief the inter-working between components for Fault-management and Failure-detection, and the Resilience and Monitoring Frameworks of ANA [36][37]. This is shown on the architecture meant for the ANA node, which is derived from the generic architecture of the Unified Implementation Framework to be published by the International Journal of Network Management, John Wiley & Sons [35].

# Table of Contents

# 1 INTRODUCTION

*Role/relevance of the deliverable/work for the overall ANA project*

It is in the area of Network Management where traditionally, a lot of objectives and operations related to the operation and maintenance of networks and systems still require human intervention. Complexity in managing systems and networks is on the increase owing to advanced communication services and complex protocols, hence a call to *task automations*, which in turn caused the birth of *autonomics*. Management functions are already known [1] and novel management functions are being researched on, and so the issue now, as in ANA, is how to make management functions such as fault-management functions autonomic. For ANA, like in all clean-slate types of projects there is a call for open frameworks for implementing self-managing (autonomic) functions across each of the traditional FCAPS management planes. This deliverable presents one such management related framework that focuses on Fault-Management and Failure Detection issues.

## 1.1 Scope of Deliverable

This is a first draft that is based on our work on defining a **Unified Framework for Implementing Autonomic Fault-Management and Failure-Detection for Self-Managing Networks**, a framework that can be applied to ANA or similar projects related to autonomic networking. In the upcoming project years, the framework will be followed in implementing for ANA as a case study. The architecture is yet to be refined and interfaces are yet to be further defined in greater detail. Experiences gained in following the framework in implementing some selected components and mechanisms for ANA will be communicated to the wider community via Publications. This will help in evolving the Unified Implementation Framework to be published by the International Journal of Network Management, John Wiley & Sons [35].

## 1.2 Structure of the document

This document is organized as follows: Section 2 introduces terminology and definitions. Section 3 presents parts of the **unified framework for implementing autonomic fault-management and failure-detection.** Section 4 captures and defines the key components (functional blocks) that ought to play a role in autonomic fault-management and failure detection for self-managing networks. Section 5 presents an architecture meant for implementation in the ANA node that is derived from the generic architecture of the unified framework in [35]. In both section 4 and section 5, we also present some problem statements and give propositions on how to approach or solve the problem in question, marked in the text "**Proposition:**", as well as capturing some requirements that designers and implementers of functional blocks of an autonomic node must take into account. The requirements are marked "*Requirement*". Section 6 gives a conclusion and insight into further work. **Appendix A** provides more detailed supplementary information.

# 2 TERMINOLOGY

## 2.1 Abbreviations

ANA - Autonomic Network Architecture

FCAPS – Framework for **F**ault-Management, **C**onfiguration-Management, **A**ccounting-Management, **P**erformance-Management, **S**ecurity Management

TMN – Telecommunication Management Network

NMS – Network Management System

ANM – Autonomic Node Manager

FB - Functional Block

FDE – Failure Detection Engine

CDE – Challenge Detection Engine

IDE – Incident Information Dissemination Engine

RE - Resilience Engine

MOF – Meta-Object Facility

IDP - Information Dispatch Point

MFB - Monitoring Functional Block

ODM - On-Demand Monitoring

IC - Information Channel

## 2.2 Definitions

**Autonomic Fault-Management and Failure-Detection** - We talk about autonomic fault-management and failure-detection when there are co-operative and collaborative mechanisms implemented in nodes and the network as a whole, to automatically detect faults, errors and failures, share knowledge about such incidents among entities, diagnose or localize faults, as well as removing faults throughout the operation lifetime of a node and the network as a whole.

# 3 IN PURSUIT OF A UNIFIED FRAMEWORK FOR IMPLEMENTING AUTONOMIC FAULT-MANAGEMENT AND FAILURE-DETECTION FOR SELF-MANAGING NETWORKS

Traditionally network management is divided into five so called management planes namely: **F**ault-management, **C**onfiguration-management, **A**ccounting-management, **P**erformance-management and **S**ecurity-management. This is known as the FCAPS management Framework [1]. Fault-management, the focus of this document, deals with the following functions: *fault-detection, fault-diagnosis, localization or isolation, and fault-removal*, whereby fault-diagnosis or fault-localisation or fault-isolation is the process of deducing the exact source/cause of an error or failure from the set of observed error/failure indications [5]. A *fault* is the cause of an *error* [10]. It is the physical or algorithmic cause of a malfunction [2]. *Faults* manifest themselves as *errors* [2]. An *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 [5]. *Error* is a consequence of a *fault* [10][5]. "Faults may or may not cause one or more errors. Errors may cause deviation of a delivered service from the specified service, which is visible to the outside world. The term *failure* is used to denote this type of an error. Errors do not need to be directly corrected, and in many cases they are not visible externally. However, an error in a network device or software may cause a malfunctioning of dependent network devices or software. Thus, errors may propagate within the network causing failures of faultless hardware or software. Some faults may be directly observable, i.e., they are problems and symptoms at the same time. However, many types of faults are unobservable due to their intrinsically unobservable nature."[**] [5].

Fault-management and Failure-Detection, why split, when in actual fact Fault-management as understood in the FCAPS management framework includes failure-detection? It is because of two reasons why in this work, the issues are presented as if they were somewhat two separate but tightly related issues. The first reason is that there are cases such as in fault-tolerant system/network designs, where one can talk about faults and errors needing to be detected and handled by fault-mitigation, masking and containment methods such that errors do not cause a service failure(s) of the considered fault-tolerant system. The second reason is that the FCAPS

[**] Extract from the article: "A survey of fault localization techniques in computer networks" by M. Steinder and A.S. Sethi, Copyright Elsevier 2004. Published in the Journal – Science of Computer Programming 53 (2004) 165-194.

framework emerged from the Telecommunications world and over the past years variants of the framework have been adopted into the traditional IT systems world where it is also common to hear about research challenges and issues dedicated to the subject of failure-detection without even going further into the bigger framework of fault-management, which includes processes of removing faults (automated fault-removal or with human intervention). It is because of these two reasons that we seek to unify some issues from the traditional FCAPS framework and other worlds where issues to do with faults, errors and failures are a common subject, into some kind of unified framework towards the implementation of autonomic fault-management and failure-detection as defined later in this document and in [35]. Therefore, fault-management as presented in this document and in [35], deals with fault-detection (for those types faults for which there are mechanisms to detect them before an error occurs somewhere due to the fault having been activated), error-detection and finally fault-removal at any stage of operation of a node or network even after a failure has actually occurred and has been detected somewhere in a node or in the network.

In the areas related to fault-management and failure detection, studies show that a lot of research has been carried out and the research continues regarding mainly: mechanisms for automated fault-detection [9][24], error-detection, failure-detection [24][25][28], fault-localization or diagnosis or isolation [5][23], automated fault-removal (to some extent), fault-tolerant systems, self-healing networks [34], network resilience and survivability [7][34]. On the other hand, today's network management as known within the FCAPS management framework, is moving towards the definition and implementation of "self-managing" network functions, with the aim of eliminating or drastically reducing human intervention in some of the complex aspects or daunting tasks of network management. The big picture of self-managing systems and networks includes autonomic computing and networking [13][4][14][15][29], whereby a system (macro-level view) or even a single system function (micro-level view) is implemented with learning capabilities and the ability to autonomously, co-operatively or collaboratively with some other entities, operate on variable input sets such as monitoring data, data or information(knowledge) received from other entities, and strive to achieve and maintain some defined goals even in the face of challenging conditions. This requires a feedback control loop(s) that drive autonomic behaviours like self-adaptation and other self-* functions [4][29]. One expects that the space defined by self-managing functions of a system or network can actually grow very large depending on system or network functions that are meant to be automated. As such, *task automation* is at the very heart of self-managing systems and networks [16][4]. Central to task automation or co-operative and collaborative task automation among network entities is the aspect of *knowledge or information acquisition and sharing* among the co-operating or collaborating entities. To mention a few examples of required task automations: dynamic, adaptive or context-driven network or system configuration and performance management tasks, automated network debugging and troubleshooting tasks, automated network service auditing or scanning, and automated fault-diagnosis, etc. Decision-making processes in automated tasks are driven by knowledge or information flow among entities. The richer the acquired and shared knowledge/information, the better the decision making capability of an automated task, a self-managing (autonomic) system or network.

Now, the question is: How can the mechanisms for automated fault-detection, error-detection, failure-detection, fault-localization or diagnosis or isolation, fault-tolerance, self-healing,

resilience and survivability, automated fault-removal, and newly emerging mechanisms thereof, fit together into some *unified framework(s)* for implementing all such mechanisms as required for the needs of self-managing networks, since most of these mechanisms have been implemented in isolated technology-specific types of networks without necessarily targeting self-managing networks in the larger complex picture? We expect that self-managing networks and systems require all or some of these mechanisms in their diversity, to be implemented in the network and in nodes, together with additional co-operative and collaborative reasoning functions inside nodes which decide on which mechanism(s) to use or trigger in what context or situation. One of such required unified frameworks is a specific unified framework for implementing autonomic fault-management and failure-detection for self-managing networks. **What is autonomic fault-management and failure-detection**? We talk about autonomic fault-management and failure-detection when there are co-operative and collaborative mechanisms implemented in nodes and the network as a whole, to automatically detect faults, errors and failures, share knowledge about such incidents, diagnose or localize faults, as well as removing faults, throughout the operation lifetime of a node and the network. Such unified frameworks are required at this point in time when there are projects calling for implementing new network architectures that are flexible to support on-demand functional composition for context or situation aware networking. A number of such projects have started, under the umbrella of what are called clean-slate network designs. An example of such projects is the ANA project [33], which is looking at architectural design principles that form the foundation upon which autonomic properties of a network node can be introduced as add-ons by applying dynamic composition styles on some networking functions and algorithmic schemes.

But, before moving on to talk about frameworks for implementing autonomic fault-management and failure-detection for self-managing networks, we first give the critical issues that must be considered in order to come up with the required unified framework. It is important to point out that the unified framework we present in this document and in [35] is meant to target network nodes that are designed to be autonomic in their operations as well as autonomic networks build from such nodes. We consider that the required unified framework must take into account the following critical issues (criteria):

1. **Criterion 1:** The framework must take into account the dynamics of a self-managing node and the dynamics of a self-managing network(s) as a whole. To mention a few examples of node dynamics: faults may be activated within a self-managing node and that may result in error propagation to other nodes in the network; the self-managing node may invoke its self-healing/self-recovery mechanisms after some node-failure, and also, its triggered node-resilience mechanisms may actually fail thereby affecting the operation of other nodes that depend on its service(s); a self-managing node may resort to re-booting itself or some of its internal modules upon entering a non-recoverable state; etc. To mention a few examples of network dynamics: links, Network Interface Cards (NICs) or nodes may fail and then recover after fault-removal has been performed; network services may cease to be available for some time because of node-failures or heavy traffic loads; the mechanisms for fault-tolerance (mitigation, fault-masking, fault-containment, etc) and resilience in network systems may all fail; etc. System-on-Chips such as NICs that can self-diagnose and recover from faults are being developed [11] and such designs are required in self-managing networks. This explains why we consider that for a time-duration $[T_0\text{-}T_1]$

during which the NIC is self-diagnosing and recovering from a failure, the NIC is considered to be temporarily out of service (in a failure-state).

2. **Criterion 2:** The need to define the key components that are expected to play a role in autonomic fault-management and failure detection. A number of questions must be answered regarding the functional nature of the components, the distributed nature of the components within a self-managing node and the network as a whole; the functions related to fault-management and failure-detection that should be assigned (functions placement) to components, as well as the required interfaces between the components. There is also a need to distinguish between components, let's call them ***X*-Components**, that must be specifically defined and designed as components solely dedicated to autonomic fault-management and failure-detection related functions, and those components that are simply normal functional entities inside a node or the network that may be required by the ***X*-Components** to co-operate or collaborate in some way during the execution of some fault-management and failure-detection related functions. This actually means the *functional requirements* imposed by the ***X*-Components** on the normal components and functions of nodes must be captured and specified.

3. **Criterion 3:** Evolvability of all the components described in **criterion 2**.

4. **Criterion 4:** Traditional "self-healing" properties relate to restoration and protection mechanisms found in protocols such as SDH, ATM, MPLS, etc. The vision of implementing fully autonomic nodes and networks is that there is a need to implement self-healing as a capability of the whole node, meaning that a node in an autonomic network should exercise reloading of faulty entities within itself, deciding and starting backup components, performing fault-removal whenever possible, and overseeing the overall health of the node throughout its operation lifetime. The node should also be able to initiate network-wide collaborative troubleshooting and network debugging functions by inviting other nodes to participate in collaborative distributed fault-diagnosis/isolation when it detects errors/failures.

5. **Criterion 5:** Entities such as nodes or NICs recovering from failures should be able to announce their "*back-to-life*" status to the hosting node or to the network. For example, a server may temporarily suffer some failure, which aught to cause the hosting node to self-diagnose and self-heal, and upon successful recovery, the node then informs the network that the node itself or its hosted service/server that previously failed is back to life.

6. **Criterion 6:** In a self-managing node or network, fault-management must be co-operatively and/or collaboratively handled by a number of functional entities through the sharing of "*knowledge*" or "*information*" about failures, errors, faults, their points of occurrence or manifestations, their causality relationships, dependency relationships between entities (protocols, services, nodes, etc.) and context information etc. Functional entities have to use all such knowledge in order to execute or co-operatively request each other to execute some *assigned fault-management related functions*. In an autonomic node or network, there is a need for functional

entities to co-operate and/or collaborate in fault-detection, error-detection, failure-detection and fault-localization since one entity may be able to detect an incident within a time frame shorter that other entities can detect it (if at all).

7. **Criterion 7:** The *meta-models* required for creating (instantiating) information/knowledge *models*, as well as the *storage repositories* for information/knowledge required in order to automate knowledge-sharing, *fault-isolation*, *fault-removal* and *self-healing* processes. A meta-model captures *concepts and their relationships, as well as constraints* between those concepts and relationships. *Models* are the instances of the *meta-model*. Meta-models and models allow for the development of algorithms such as fault-diagnosis algorithms that operate on models.

8. **Criterion 8:** Support for learning capabilities that enable autonomic nodes to learn about faults, errors and failure related incidents in the network in order to enable the nodes to self-adapt or avoid sending traffic unnecessarily into the network, destined for nodes or services hosted by nodes that are considered to be in a failure state unless information about their *back-to-life* status has been received.

9. **Criterion 9:** The unified framework must be generic, to abstract from specific network protocol and architectural technologies, as well as abstracting from algorithms, methods and protocols for automated fault-detection, error-detection, failure-detection, fault-diagnosis, fault-removal, and information/knowledge sharing among network entities. As such, the unified framework must provide an implementation architecture that gives a clear picture on functions placement and algorithm placement within some specified functional blocks without going deep into or exhausting algorithmic issues. The architecture must allow for functional and mechanism diversity to allow multiple functions, multiple mechanisms or algorithmic schemes such as information/knowledge dissemination schemes or methods to be flexibly implemented or extended, and to be selectively invoked depending on context or situation as determined by an autonomic node.

In this document and in [35] we present a framework for implementing autonomic fault-management and failure-detection for self-managing networks. The unified framework we present in [35], takes into consideration the criteria we outlined above. When talking about failures and faults, we are referring to failures and faults that are encountered during the operation of a system or network. In this unified framework we talk about specialized software components for autonomic fault-management and failure-detection, functions placement (assignment), as well as the required interfaces between the components. We denote such specialized components, as well as the normal software components of a node, as Functional Blocks (FBs). The terminology used in the general FCAPS management framework [3][1] is that of so-called Function Blocks(FBs) where a Function Block is the smallest deployable unit of TMN management functionality that is subject to standardization. In our case, we try to take into account any functions in general, to cover any type of networking related functions. Therefore, we use the term Functional Block instead of Function Block, whereby, a Functional Block (FB)

means a deployable software component that has at least one function assigned during its design, such that the implemented function(s) is perceived by other functional blocks as offering some service(s) via some service access point(s) available at some interface(s) of the functional block. Therefore, in our unified framework all the types of components described in **criterion 2** above will synonymously be referred to as Functional Blocks (FBs) throughout this document.

**What has been achieved so far in fault-management and failure-detection?** As already pointed out earlier, studies show that a lot of research has been carried out and the research continues regarding mainly: mechanisms for fault-detection, fault-localization or diagnosis or isolation, fault-tolerance, self-healing, resilience and survivability, and to some extent fault-removal. What is missing now are unified implementation oriented frameworks for implementing these mechanisms for the emergent and future self-managing networks, since most of these mechanisms have been implemented in isolated technology-specific types of networks without necessarily targeting self-managing networks in the larger complex picture. The unified frameworks must take into account the nine criteria we outlined above as well as taking into account those management functions and practices defined by today's FCAPS network management framework that should go autonomic. A lot of management functions still involve human intervention. Functions such as fault-isolation (diagnosis), troubleshooting and alarm co-relations and analysis still involve significant human involvement even though there are tools to aid network management personnel sitting in an NMC (Network Management Center) or a NOC (Network Operations Center). In comparison to other frameworks such as [17][32][27], the framework we present here is broader and is strongly based on the nine criteria outlined above. The other important thing is, the unified framework we present here is meant to also accommodate clean-slate designs such as ANA [33] that aim at designing and integrating node components and functions that constitute fundamental ingredients for engineering autonomicity in nodes and across networks.

Section 4 captures and defines the key components (functional blocks) that ought to play a role in autonomic fault-management and failure detection for self-managing networks. Section 5 presents an architecture meant for implementation in the ANA node that is derived from the generic architecture of the unified framework in [35] and following the components and concepts captured in section 4. In both section 4 and section 5, we also present some problem statements and give propositions on how to approach or solve the problem in question, marked in the text "**Proposition:**", as well as capturing some requirements that designers and implementers of functional blocks of an autonomic node must take into account. The requirements are marked "*Requirement*". Section 6 gives a conclusion and insight into further work.

# 4 DEFINING THE KEY COMPONENTS THAT PLAY A ROLE IN AUTONOMIC FAULT-MANAGEMENT AND FAILURE-DETECTION

In this section, we capture and define the key components (functional blocks) that ought to play a role in autonomic fault-management and failure detection for self-managing networks. We also discuss about the distributed nature of the components within a single self-managing node or a network as a whole, the roles of each of the components, as well as interfaces required between the components. The other issues we discuss here are functions related to fault-management that we can assign among the components we define. We capture and define the required components (functional blocks), by the roles (functions) that the functional blocks must perform. Therefore, in this section and in section 5, we present a framework on how to implement autonomic fault-management and failure-detection for self-managing networks, and give a particular focus for the ANA architecture in section 5.

## 4.1 AUTOMATED FAULT-DETECTION, ERROR-DETECTION AND FAILURE-DETECTION

In the human body, the autonomic nervous system relies heavily on information flow from sensors distributed throughout the body to the decision-making components (the brain) and finally to effectors (muscles, glands, etc) that then perform an action [18]. The best way to effectively implement rich autonomic behaviours in nodes and the network as a whole is to require that entities in an autonomic node or network co-operate in exposing or sharing knowledge (information) about faults, errors and failures and their points or interfaces of occurrence or manifestation. Such knowledge must flow to those components that require the knowledge and are able to act upon it. In **Figure 1**, we present the models of inter-component interactions that allow us to capture the aspect of co-operation in fault, error and failure detection as well as the following notions: service-access points, service failure, crash-failure, points and incidents of fault/error/failure detection, viewpoints about faults/errors/failures. In these models, the notions of “request” and “response” primitives of a service-access point are generic and also include the four abstract service primitives (request, response, indicate, and confirm) associated with OSI protocol interface models.

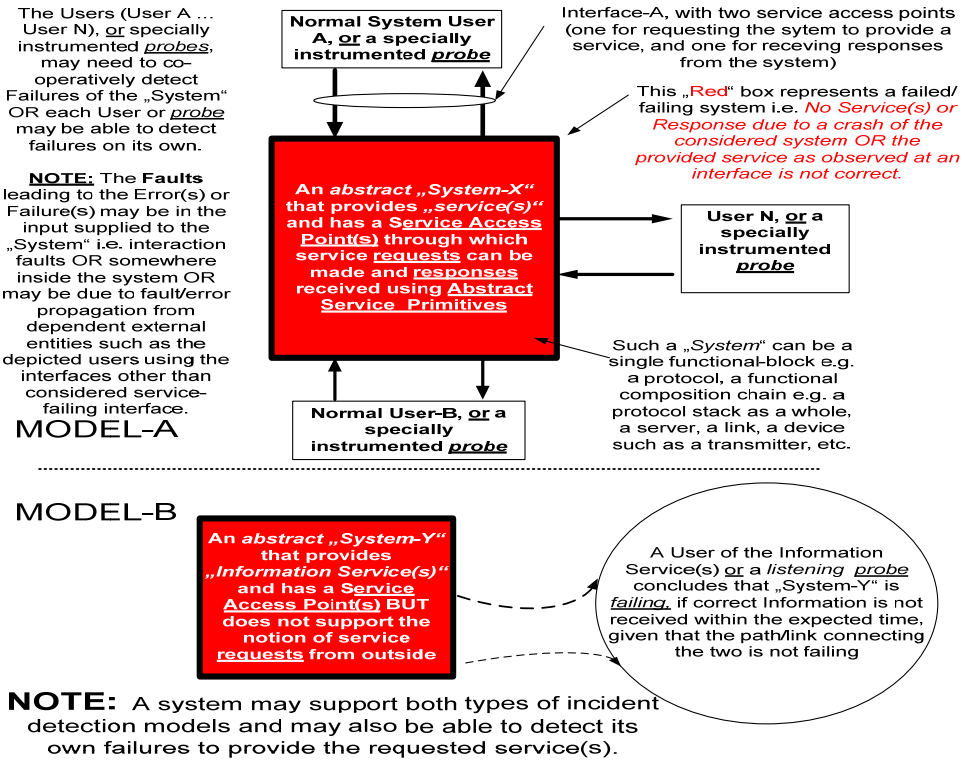


**Figure 1: The notions of: service-access points, service failure, crash-failure, points and incidents of fault/error/failure detection and viewpoints**.

We seek to answer the question: How should the tasks of automated co-operative fault-detection, error-detection and failure-detection, be achieved in an autonomic node or network?

"Some faults may be directly observable, i.e., they are problems and symptoms at the same time. However, many types of faults are unobservable due to their intrinsically unobservable nature."** [5].
**Proposition:** Any entity in the network, in a node or in a network interface card (NIC) that has the ability to automatically observe any of those types of faults that are directly observable is potentially a *fault-detector*. For example, the firmware of a network interface card may be designed with mechanisms to detect the "out-of-sync" fault described in **Figure 2**, possibly in co-operation with the main CPU and clock of the router, or some other mechanisms to detect this fault may be implemented elsewhere. In general, *fault-detection* is understood as a process of capturing on-line indications of network disorder provided by malfunctioning devices in the form of indications such as alarms [5][34]. However, in some cases faults i.e. root causes of malfunctions are only revealed by thorough study of network and protocol interaction behaviours by human experts, resulting in classification of the root causes as faults that need to be detected

** Extract from the article: "A survey of fault localization techniques in computer networks" by M. Steinder and A.S. Sethi, Copyright Elsevier 2004. Published in the Journal – Science of Computer Programming 53 (2004) 165-194.

in order to employ counter measures, sometimes even without advocating for changes to a standard-conformant protocol whose normal behaviour is the one activating faults.

**Proposition:** Because *Faults* manifest themselves as *errors* [2] and because some faults are not directly observable(detectable), we say any entity inside a node or in the network which is able to *detect errors* (errors internal or external to the detecting entity e.g. an error propagated by another entity) is potentially an *error-detector*. Such an error-detector can be any functional block inside a node that interacts with some other functional blocks e.g. an application, a server software component, a routing or a forwarding functional block. For those types of faults that can not be directly detected *error-detectors* serve to indicate the presence or activation of a fault somewhere in a node or the network.

*External failure manifestations* (*failure-symptoms*) and *failure-indications* in form of externally visible errors or lack of a response within some time constraints or some other deciding factors, may appear at different points and interfaces of a component/system in the network. Any entity inside a node or in the network which is able to capture such an *event* is potentially a *failure-detector*. Such a failure-detector can be a functional block that uses a service(s) provided by another functional block through the service-provider's service access point. An *event* is an exceptional condition occurring in the operation of hardware or software/component [5]. Failures may relate to other entities outside the failure-detecting entity or may relate to the failure-detector itself, meaning that an entity may be able to detect its own failures as well as internal errors. *Failure-symptoms* are observed as *alarms*— notifications of a potential failure [5]. As such, every functional block that interacts with another functional block ought to serve as potential failure detector (sensor) apart from the purpose it was designed for.

Therefore, given the notion of a "service" provided by a functional block, or a function implemented by the functional block, we have a notion of *service-failure* or *function-failure* given by so called *failure-modes*, which represent availability or performance problems pertaining to the service or function [7][10]. In [7] the following examples of failure-modes (*F1…F4*) are defined.

- *F1: service/function ceases to exist (e.g., a cable connection is broken),*
- *F2: service/function introduces unacceptable delay (e.g., one of the host-to-host links is congested),*
- *F3: service/function produces erroneous output (e.g., bit errors are introduced in a link between routers),*
- *F4: service/function occasionally does not produce output (e.g., packets are lost due to buffer overflow).*

Actually, failure modes can be more than one, depending on the requirements imposed on the service or function by its users, i.e. our potential failure-detectors of the service or function.

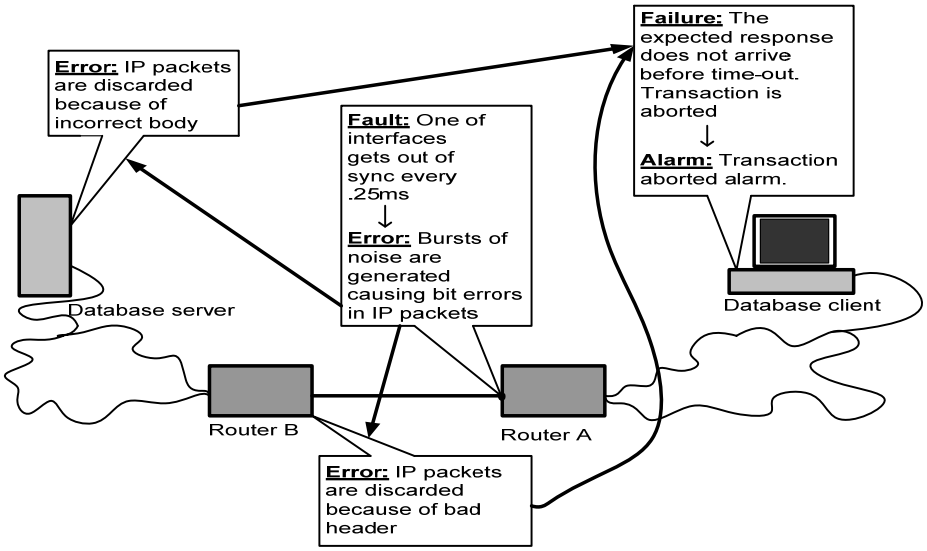


**Figure 2**: **Distinction between fault, error, failure, and symptom[**] [5]**

## 4.2 AUTOMATED ALARM-GENERATION

An *alarm* is notification of a specific event. It may or may not represent an error [2]**.** Alarms are specific types of notifications concerning detected faults or abnormal conditions. Generally speaking, today's traditional network management frameworks already include automated alarm generation e.g. threshold-crossing related alarms, etc. For example, SNMP supports *trap* and *inform* type of notification messages. **Proposition:** Since extensive and rich knowledge sharing is key to implementing autonomic network-wide collaborative behaviours regarding self-adapting behaviours, an alarm generating functional block must be designed with extra reasoning capability to reason about the entities in the network which may be interested in receiving knowledge about the alarm(s) and request the network to disseminate alarm(s) information accordingly. When designing functional blocks, the designer should ensure that a functional block that generates an alarm(s) includes in the generated alarms, information that will help with determining the cause of the potentially abnormal situation, including information identifying the alarm generating entity, and other information related to side effects as pointed out in [2].

## 4.3 AUTOMATED FAULT, ERROR, FAILURE AND ALARM INFORMATION DISSEMINATION

Reliable and timely dissemination of knowledge about detected faults, errors, and failures, as well as alarm information is required in an autonomic node or network. Traditionally software entities such as protocols may be designed with some inherent error reporting mechanisms via the use of error-codes in the messages exchanged during communication with some other entity or entities. Other error-reporting mechanisms rely on specially designed error-reporting protocols such as ICMP. Both types of error reporting mechanisms are used in today's networks, which do not need to convey a lot of information among nodes. For efficient, timely, robust and reliable

[**] Figure taken from the article: "A survey of fault localization techniques in computer networks" by M. Steinder and A.S. Sethi, Copyright Elsevier 2004. Published in the Journal – Science of Computer Programming 53 (2004) 165-194.

information dissemination, the potentially huge and diverse information sets together with the associated diverse information recipients sets, as well as information delivery requirements and constraints expected for an autonomic network, diverse information/knowledge dissemination mechanisms may be required in different situations. Moreover, the designer of a functional block that may need to disseminate information about detected faults/errors/failures, such as a routing functional block or even an application, does not need to think about the selection of a particular information dissemination mechanism suitable for a particular situation or environment during the operation of the functional block. **Proposition:** We propose to implement a *generic information dissemination service e.g. an information dissemination structure e.g. a bus* inside an autonomic node such that any functional block that generates information and "reasons" about the recipient entities need only influence the *information dissemination service* to select the most efficient, timely and reliable information dissemination mechanism or protocol as may be required by the requesting functional block and disseminate the information/knowledge accordingly. Functional blocks acting the role of information/knowledge receivers need to register with the *dissemination service* in order to receive information or event notifications.

## 4.4 THE "SPINAL CHORD" CONCEPT FOR INFORMATION DISSEMINATION

**Proposition:** For information/knowledge sharing among network nodes, we propose a concept of what we call a spinal cord, inspired from the spinal cord of a human. The spinal cord of a human is a structure that consists of specialized channels for the following functions: conducting sensory information from the peripheral nervous system (both somatic and autonomic) to the brain; conducting motor information from the brain to our various effectors such as muscles and glands; and serving as a minor reflex center.

**In the case of an autonomic network:** Certain kinds of information/knowledge such as information related to fault/error/failure notifications among autonomic nodes of the network, as well as some monitoring data are very sensitive. Delaying the flow of such information (or data) or losing them can severely geopardize the autonomic operation of the network. Therefore, we define the Spinal Cord of an Autonomic Network (SCAN) as a structure consisting of network-wide information channels through which specialized types of messages/information that are especially sensitive to delay or loss can be conveyed efficiently. In an autonomic network, the special information channels constituting the spinal cord (SCAN) must be created and managed by the Autonomic Node Managers (described in Section 4.5 below) of the autonomic nodes. Each Autonomic Node Manager (ANM) of a node ensures that, the *spinal channels* it creates together with a neighbour ANM are monitored to ensure that the channels are always available for transmitting time-critical messages/information. As such, the whole spinal cord is monitored to ensure certain properties like very low message delay, and must be secured from malicious activities. The dynamics of the spinal cord involve the creation of new channels in case of failures to ensure availability, resilience and connectivity of a node to the spinal cord. Just like in TMN [2], nodes may have some interfaces (NICs) dedicated to management related communications and information flow. Such dedicated interfaces should be used for establishing the SCAN.

## 4.5 AUTOMATED AND COLLABORATIVE FAULT-DIAGNOSIS, LOCALIZATION OR ISOLATION

*Fault-Isolation* (also referred to as *Fault-Diagnosis* or *Fault-localization*) is the process of deducing the exact source/cause of an error or failure from the set of observed error/failure indications [5]. Automating this process as well as the process of fault/error/failure detection and fault-removal is paramount to enabling autonomicity and resilience of a system, component or the network. "In a large network, fault-localization should be performed in a distributed fashion."[**] [5].

How about fault-localization in an autonomic node or network? The process of fault-isolation can be initiated by an *error-detector* or a *failure-detector* and completed by this single entity or can require the co-operation and or collaboration of multiple entities to complete the process. Fault-isolation can also be initiated by some other type of an entity (not the entity directly detecting the error/failure) e.g. the traditionally known *Network Management System* (NMS) or a *node manager* (implemented within a node) that must first become aware of the error or failure detection event and then gathers all the required information for diagnosis from all the relevant entities and performs the fault-isolation process. The question of where to initiate or implement a fault-isolation procedure, whether in the error-detecting entity (*error-detector*) or failure-detecting entity (*failure-detector*) or having a partial implementation of the mechanisms/procedures in the detecting entities that is complemented by additional collaborative mechanisms/procedures implemented in some specially designed entities dedicated to fault-isolation such as the NMS or a node-manager, depends on the type of faults being addressed and many other factors such as design complexity, complexity of causality relationships among faults, the overhead that would be imposed on the detecting-entity in the case that the entity has to continue to provide services in the presence of the manifesting faults (i.e. is fault-tolerant) while at the same participating in a fault-isolation process. Traditional NMSs have some degree of being autonomic in some aspects such as taking decisions on adaptive network configuration or reconfiguration, etc. The node manager described above can also be implemented as an autonomic entity that implements a number of autonomic functions such as mechanisms to detect corrupted local entities e.g. memory, crushed processes, mechanisms to perform dynamic upgrading of entities/services without service disruption, garbage collection, reloading of modules and functional blocks, replacing some corrupted entities such as modules, rebooting the node when determined as the last resort to recover from a Byzantine state, as well as overseeing the health and operation of the node. **Proposition:** We propose that every autonomic node implement an **Autonomic Node Manager** (**ANM**) that makes decisions and performs all such autonomic functions. It is because of this **Autonomic Node Manager** (ANM) that a node can be said to be an "autonomic node". The question is on how to use diverse fault-localization techniques/methods in autonomic nodes and networks, meaning, how to assign them in their diversity to normal functional blocks or *specially designed fault-diagnosis components* of an autonomic node. [5] provides a summary of fault-localization techniques ranging from 1) **Artificial Intelligence techniques** such as rule-based systems, neural networks, model-based systems, decision trees, and case-based systems; 2) **Model traversing techniques**; 3) **Fault**

[**] Extract from the article: "A survey of fault localization techniques in computer networks" by M. Steinder and A.S. Sethi, Copyright Elsevier 2004. Published in the Journal – Science of Computer Programming 53 (2004) 165-194.

**propagation models** such as code-based techniques, dependency graphs, Bayesian networks, causality graphs, and phrase structured grammars.

## 4.6 AUTOMATED ALARM PROCESSING

Alarm processing means acting upon an alarm received from an alarm generating entity. Traditionally, alarms generated by a network element such as a switch or router are sent to the Network Management System(NMS) where the alarm information is stored and some limited alarm processing is done by the NMS. The network operations personnel still play some role in alarm filtering and correlation, and eventually in fixing some problems i.e. removing faults. How about alarm processing in an autonomic node or network? What sort of entities (functional blocks) should process (act upon) alarms? Of course all this depends on the type of alarms and the intended recipients of the alarm. Clearly a functional block such as a routing functional block inside an autonomic node may need to generate an alarm related to say an overloaded path or link, which is intended to be received by other routing functional blocks in other nodes, in order to trigger self-adaptation behaviours across routing functional blocks of some scope. **Proposition:** There are some types of alarms that must be generated by the autonomic node's functional blocks and need to be examined (processed) by the node's Autonomic Node Manager (ANM), which may decide to trigger complex self-adaptation behaviours for the node as a whole or to shut down certain functions temporarily. Because alarms are specific types of notifications concerning detected faults or abnormal conditions, some *specially designed fault-diagnosis components* of the autonomic node need to act upon or to use alarm information related to fault manifestation and abnormal conditions in executing the automated fault-diagnosis processes.

## 4.7 AUTOMATED FAULT-REMOVAL

*Fault-Removal* involves the reduction of the presence (number, seriousness) of faults [10]. Again, the questions we need to answer are about the entities in an autonomic node or network that can initiate a fault-removal process, as well as the co-operative and or collaborative fault-removal procedures and processes.

**Proposition:** The process of fault-removal can be initiated by a *fault-detector*, an *error-detector* or a *failure-detector* and completed by this single entity or can require the co-operation and or collaboration of multiple entities to complete the process. Fault-Removal can also be initiated by some other type of an entity (not the entity directly detecting the fault/error/failure) e.g. the traditionally known *Network Management System (NMS)* or a *node manager* (resident within a node) that either removes the faulty entity from the communication picture, reboots a faulty entity, reloads a faulty module or replaces the faulty entity etc. The question of where to initiate or implement a fault-removal procedure, whether in the fault-detecting entity (*fault-detector*), error-detecting-entity(*error-detector*) or failure-detecting entity (*failure-detector*) or having a partial implementation of the mechanisms/procedures in the detecting entities that is complemented by additional collaborative mechanisms/procedures implemented in some specially designed entities dedicated to exercising fault-removal such as the NMS or node-manager, depends on the type of faults being addressed and many other factors such as design complexity, complexity of causality relationships among faults, the overhead that would be imposed on the detecting-entity in the case that the entity has to continue to provide services in the presence of the manifesting faults (i.e. is fault-tolerant) while at the same participating in a

fault-removal process. As already mentioned, we propose to implement an **Autonomic Node Manager** (**ANM**) in an autonomic node.

Because not every automated fault-isolation process completes successfully, not every root-cause (fault) can be identified, and not every fault-removal process completes successfully, the Autonomic Node Manager must provide history of all attempts to achieve some goals through logs, which can be inspected by some human experts when necessary.

## 4.8 THE REQUIREMENT FOR META-MODELS FOR CREATING MODELS AS WELL AS THE REPOSITORIES FOR INFORMATION OR KNOWLEDGE REQUIRED IN AUTOMATED FAULT/ERROR/FAILURE-DETECTION, FAULT-ISOLATION, FAULT-REMOVAL, AND SELF-HEALING PROCESSES

In this section we present the role of information (*knowledge*) meta-models, models and model storage repositories required in automated fault-diagnosis, fault-removal and self-healing processes. We also answer the questions as to the nature of the knowledge, the functional blocks that must automatically create this knowledge and store it into knowledge repositories, as well as the functional blocks that then interact with the repositories to fetch the knowledge and use it. As we present the ideas, we also mark the *design and functional requirements* imposed on the general or normal functional blocks of an autonomic node e.g. a routing functional block or an application, to enable co-operation in knowledge creation and sharing.

One of the key assets required for communicating information/knowledge to some entities or the sharing of information/knowledge among entities e.g. functional blocks within an autonomic node or functional blocks residing in different autonomic nodes are information/knowledge *models*, which are instances of an information/knowledge *meta-model*. As already mentioned earlier, knowledge/information about failures, errors, faults, their points of manifestations, their causality relationships, dependency relationships between entities (protocols, services, nodes, etc.), context information, etc must be shared among functional blocks that can act upon it. A meta-model captures *concepts and their relationships, as well as constraints* between those concepts and relationships. Models are the instances of the meta-model. The meta-model is also useful for designing the repository for storing the models and for allowing any entity which "understands" the meta-model to instantiate the meta-model by creating a model and or read models from the repository. Such repositories must be implemented in an autonomic node. A number of meta-models are needed in order to implement repositories e.g. registries in autonomic nodes and to allow functional blocks of an autonomic node to instantiate the meta-models and store information/knowledge models as required. The concepts of meta-models, models, and repositories are well known in the newly emerged model-driven systems development frameworks like the MDA (Model-Driven Architecture) [20]. In MDA, MOF (Meta-Object-Facility) language [21] is used for creating meta-models and implementing MOF-compliant meta-model specific repositories that can be used by model-instantiators to create and store models (instances of a particular meta-model), and also used by so-called model transformers to read a source model, transform it with the aid of a "*model-walker*" and store the newly created model into a target repository. We propose to exploit all these concepts beyond the notion of MDA tools for systems development, to use them in the context of knowledge creation,

knowledge-flow and knowledge sharing in autonomic processes. Below, we present the meta-models we identified as required in automated fault-isolation, fault-removal and self-healing processes within an autonomic node or collaboratively across an autonomic network, namely:

- the "***Dependability Meta-Model***",
- the "***Faults-Errors-Failures Causality Meta-Model***",
- the "***Detected Faults-Errors-Failures Meta-Model***",
- the "***Alarm Information Meta-Model***",
- and the "***Meta-Model of KnowledgeSuppliedByMonitoringComponents***".

This means, every autonomic node in the network must implement multiple knowledge-store repositories corresponding to these five meta-models. This also means every functional block of an autonomic node e.g. a routing functional block, by design, must know some or all of these meta-models in order to instantiate some concepts and relationships and store/register the information/knowledge into a repository corresponding a specific meta-model, during the boot-up time of the functional block and/or during run-time. **Proposition:** We propose to use MOF to specify the meta-models and implement the repositories. One of the advantages of using MOF-based repositories versus using relational databases is that MOF-based repositories can support model serialization to say XMI [22], storage of multiple models, model checking, validation and versioning, event notifications from the repository, as well as allowing an entity running as a client of the repository e.g. a *Fault-Diagnoser Entity* to import a model into its memory space and invoke a model-walker that was compiled and linked as part of the entity's executable, in order to reason and operate on the model. Meta-models, model storage, model importing and exporting, model walking, transformation or creation of new models, all facilitate the development of complex algorithms required in task automation in processes such as information creation and sharing, and fault-diagnosis, required in self-managing networks. All these meta-models must be designed way before functional blocks of the autonomic node are designed.

The **Meta-Models**, their significance and roles are elaborated in **Appendix A**.

## 4.9 PUTTING THE PIECES TOGETHER

Now that we have a picture on the components (functional blocks) which play a role in autonomic fault-management and failure-detection namely *fault-detectors, error-detectors, failure-detectors, knowledge repositories e.g. registries*, and the *Autonomic Node Managers* (ANMs), we seek to identify design principles that allow all these components, resident in an autonomic node, to co-operatively or collaboratively address fault-management and failure-detection issues of the node and at the same time collaboratively together with components of similar functionality residing in other autonomic nodes, address fault-management and failure-detection issues of the autonomic network. Every functional block of an autonomic node can either play the role of a *fault-detector, error-detector* or a *failure-detector*, and may possibly implement some kind of fault-isolation and fault-removal procedures as discussed earlier. Because of the problem of additional overhead in such cases, in attempting to perform fault-isolation and fault-removal, and because of the complexity that may be involved in fault-isolation and fault-removal, especially in the case in which distributed collaboration among entities is required, the normal functional blocks of the autonomic node need to delegate the more complex and network-wide tasks of fault-isolation, distributed failure-notifications, and fault/error/failure/alarm information dissemination, etc, to specially designed and dedicated autonomous functional blocks in which "reasoning" and more complex tasks related to node-

local and distributed collaborative fault-management need to be handled in an intelligent manner with regards to say, resource utilisation, creation of special information channels and information/knowledge flow management within the node and across distributed network entities. **Proposition:** We propose to call the collective set of the *specially designed dedicated autonomous functional blocks of an autonomic node which are solely dedicated to the more complex functions of fault-management, detection of failures of node-external entities, as well as the dissemination of information/knowledge to some entities in the network*, the **F**ailure **D**etection **E**ngine (**FDE**). It is because of the relation that "*an activated fault may lead to an error that may lead to a failure that may even further lead to a fault and so on and so on"*, and because a failure is the final extreme problem that should be avoided, that we call the composite dedicated functional blocks in question, the **FDE** [35]. The Autonomic Node Manager is another component dedicated to playing some roles in most of the management related functions.

Therefore, the **FDE** [35] must consist of a main functional block and a number of other functional blocks that are controlled by the main functional block of the FDE. The main functional block of the FDE must implement some autonomous reasoning functions that allow the FDE to run as a service during the operation of the autonomic node. The rest of the functional blocks of the FDE must implement some specific functions related to fault-management, incident-information dissemination, learning about local incidents related to faults/errors/failures/alarms as well as the incidents external to the node, and triggering those failure-detection mechanisms such as special monitoring/probing mechanisms that enable interested node-local entities to get notified of external failures, as well as other functions that may need to be implemented within the realm of the FDE.

Functional blocks acting as fault-detectors, error detectors or failure detectors need to supply information/knowledge about detected faults, errors and failures to the special dedicated functional blocks (i.e. the FDE) of the autonomic node which are meant to be the *central consolidation point where knowledge/information about local detected service-failures, errors and faults are registered and can be cleared upon recovery and repair.* The information about detected faults, errors and failures, including the causes of failures if known, form vital knowledge that *should be shared* with interested entities in the autonomic node or in the network in order to enable self-adaptation behaviours, fault-removal and self-healing(self-repair). The FDE sees all potential incident detectors i.e. the functional blocks of the node, as if they were sensors that supply it with knowledge by registering the knowledge, following the "*Detected Faults-Errors-Failures Meta-Model"* (described in **Appendix A**). As such, the FDE sees the task of failure detection, error-detection and fault detection as being shared across potential *failure detectors, error-detectors* and *fault-detectors* distributed in the whole node. *Requirement:* Every *fault/error/failure-detecting entity (functional block)* should reason on the impacts of the detected fault/error/failure upon the functional composition stack as a whole or upon a single remote entity (functional block), and if the functional block does not have built-in dissemination capability, it must request the FDE to disseminate the fault/error/failure-information accordingly. It does not mean that a *resilient entity* (functional block) detecting a error or failure should not attempt to *contain, mitigate* or *mask* a fault or failure. It should register knowledge regarding any detected incidents, because there are other autonomous functional blocks inside the node e.g. the *Autonomic Node Manager*, which consults the appropriate repositories of the FDE and perform garbage collection, reloading of reported faulty/failing entities and modules etc, among other management functions. Fault-diagnosis functions of the FDE also rely on such registered

knowledge and so, complex fault-diagnosis as well as information/knowledge dissemination is performed by the FDE while the more complex fault-removal and self-healing (self-repair) are performed by the Autonomic Node Manager (ANM) in co-operation with the FDE, as already described earlier. Apart from the knowledge registration functions (behaviours), the functional blocks of the autonomic node should implement functions meant for their resilience.

Therefore, the **FDE** and its **Incident Information Repositories**, the **Autonomic Node Manager (ANM)**, and **Network Management Systems (NMSs)** form the **X-Components** described in **criterion 2** in the introduction section. What all this leads us to, is the need to identify *functional requirements* and *design principles* imposed by these **X-Components** on each functional that plays a role in fault-management and failure-detection. Some of these requirements and design principles have already been discussed earlier and more requirements are discussed in section 3.

# 5 AN ARCHITECTURE FOR IMPLEMENTING AUTONOMIC FAULT-MANAGEMENT AND FAILURE-DETECTION FOR "ANA NETWORKS"

In this section, we present an architecture (see **Figures 4, 5 and 6**) derived from the generic unified framework to be published by the International Journal of Network Management, John Wiley & Sons [35]. The figure shows an autonomic node and its components for autonomic fault-management and failure detection, as well as their interaction with the general (normal) functional blocks of the node. The figures also show the interworking between components for Fault-management and Failure-detection, and the Resilience [36] and Monitoring [37] Frameworks of ANA.

## 5.1 THE DESCRIPTION OF THE COMPONENTS AND THEIR INTERFACES

This sub-section describes the specialized components of the architecture and interfaces between the components, as well as interfaces used by the general (normal-purpose) functional blocks of the autonomic node, such as a routing functional block or an application (see **Figures 5 and 6**). For the ANA architecture, the FDE defined in the generic framework [35] is split into two components, namely a Challenge Detection Engine (CDE) and an Incident Dissemination Engine (IDE). It is understandable to say that incidents such as fault activations, errors, failures can be classified as "challenges" to the operation of a system or the network as a whole, hence the name Challenge Detection Engine. The Incident Dissemination Engine (IDE) is meant for the dissemination of incident information to entities distributed in the network. A number of instances of specialized Challenge Detection Engines may be implemented in a system. Some may be specific to the detection of security threats since security threats can be classified as challenges to the operation of the system. The type of a CDE we describe here is specific to faults, errors, failures and alarms, and is shown expanded, to see its composite nature and its internal components.

The figure below (**Figure 4**) is a high level view of the key components for autonomic fault-management and failure detection. This view excludes the view of the general normal-purpose functional blocks of the node e.g. a routing functional block, which also play some roles in the co-operative processes of autonomic fault-management and failure detection as described in **section 4**. The view of the general normal-purpose functional blocks of the node, again only with respect to autonomic fault-management and failure detection, is shown on **Figure 5**. The functions and interactions between the ANM and the other depicted components are briefly described in **sub-section 5.1.5**. The *dotted interactions* between the ANM and the other components are there to indicate that the ANM is the one that starts and manages the execution of these engines as explained later in **sub-section 5.1.5**. The interactions between the

components, represented by the *solid lines*, may be bi-directional, synchronous or asynchronous types of communication, either as procedure calls or information passing/flow by push or pull models. In this document, we do not seek to extensively provide the full functionality and capabilities of the ANM, but rather to focus on the role of the ANM in relation to fault-management, failure-detection and resilience issues. The Resilience Engine (RE) is described in more detail in [36]. The role of a *Functional Composition Framework or Logic* is described in detail in **Appendix A** in the sub-section: the **Dependability Meta-Model**. The DMRepo (Repository) is described in **sub-section 5.1.4**.

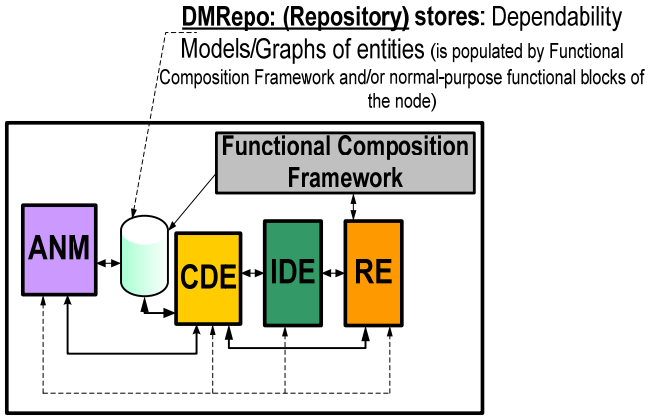


**Figure 4: The Architecture for ANA (a high-level view)**

**Figure 5** shows the view of the general normal-purpose functional blocks of the node e.g. a routing functional block, as well as the view of the monitoring framework [37], again with respect to co-operative processes for autonomic fault-management and failure detection described in section 3.

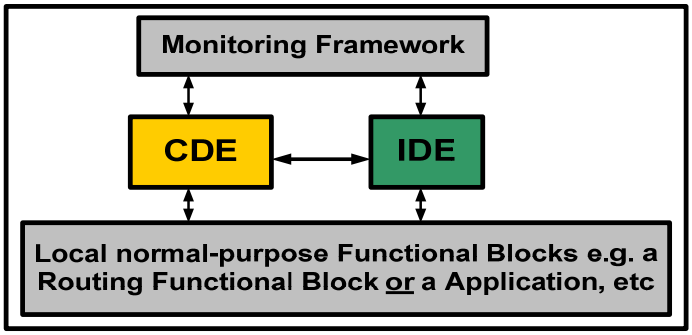


**Figure 5: The view of the Monitoring Framework and general/normal functional blocks of the node (a high-level view)**

In the next sub-sections, we describe the key components, with a particular focus on the CDE, the IDE and a more detailed view of the high level views presented by **Figures 4 and 5**. Therefore some of the interfaces presented in the high level views above, identified as required for the completeness of the bigger picture, are not specified in this document, but are rather described in brief along the description of the components, such that the interfaces can be further specified in detail as an effort of evolving the framework presented in [35].

### 5.1.2 The CDE (Challenge Detection Engine)

We start by briefly describing the components of the CDE.

1) The **core reasoning functions of the CDE**: are responsible of invoking some behaviours required at the initialisation of an autonomic node and when the node is joining/leaving the physical network of some scope (as defined by some policy), such as presence advertisements, discovery of neighbour CDEs and learning of faults/errors/failures already known by other CDEs in the network. The core reasoning functions are responsible of selecting appropriate failure detection mechanisms (e.g. special monitoring protocols/techniques) for some types of failures classified as external to the node such as path, link and remote-node-liveness failures, etc. The selection of appropriate "external-failure" detection mechanisms should be based on "survivability requirements" specified by local entities that require to be notified of such failures. **Proposition:** The survivability requirements for diverse functional blocks of the node must be captured and functional blocks must be designed in such a way that they are able to express the survivability requirements to the core reasoning functions of the CDE. Survivability requirements include information such as *time constraints* within which a functional block must be notified of the failure, to allow the functional block to react in time to the failure. An example of a survivability requirement: "*notify me within 50ms of incident detection*". The core reasoning functions are also responsible of invoking *Fault-Diagnosis/Localization/Isolation* functions including network debugging functions. This means the core reasoning functions consult the *Repositories* that store "knowledge" regarding dependability models/graphs, fault-error-failure causality graphs, and the other *Repositories e.g. Registries(Caches)* that play a role in fault-diagnosis/localization/isolation namely the "***Local Incidents Registry", "External Incidents Registry", "Asserted Incidents Registry", and the "Local and External Alarms Registry"***. The core reasoning functions may attempt to perform *fault-removal* depending on the nature of the identified fault(s), otherwise some of the fault-removal should be exercised by the *Autonomic Node Manager* of the node, which has the ability to garbage-collect failing or crashed runtime entities(processes and threads), reload modules and functional blocks, upgrade or replace some service components, etc.

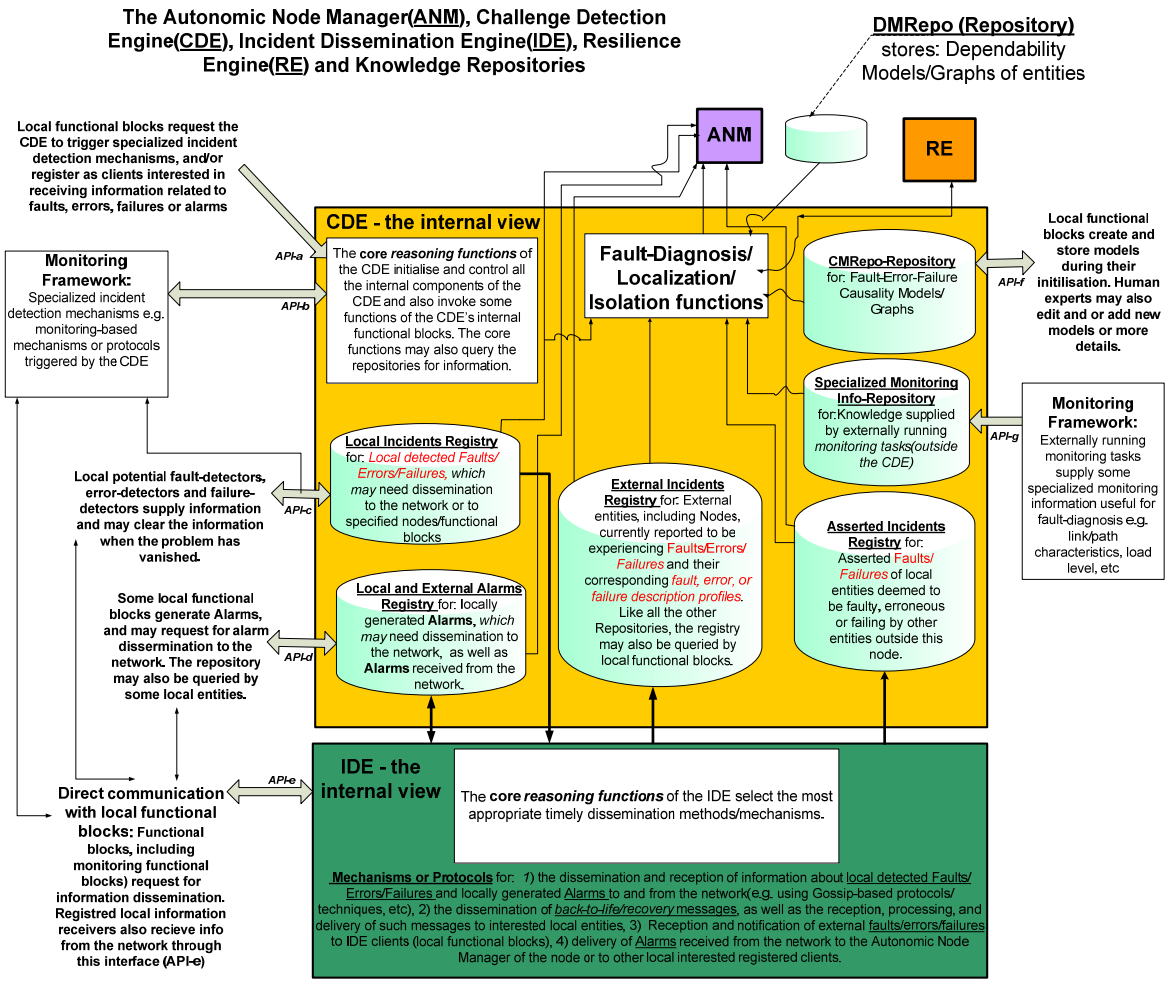


**Figure 6: The Architecture for ANA (detailed view), derived from the generic architecture in [35]**

*2)* **Interfaces (APIs) of the CDE:**

*a.* ***API-a***, used by CDE clients i.e. functional blocks to: request the CDE to trigger the mechanisms specialized for the detection of some types of failures considered external such as link, path, remote-node/service failures.

*b.* ***API-b*** is bi-directional and should implement two sets of functions: a set of functions that are called by the CDE's core reasoning functions to trigger, those specialized incident detection mechanisms which are based on specialized monitoring techniques (if not invoked yet), supply parameters to the resulting monitoring tasks and manage the monitoring tasks. This means the monitoring components required by the CDE should support the notion of customizable, programmable monitoring, monitoring-session and session management primitives such as those defined by the On-Demand Monitoring (ODM) paradigm [4][6][12]. The second set of functions is used by the specialized incident detection mechanisms such as those based on monitoring techniques, running as

CDE triggered monitoring tasks, to inform the CDE's core reasoning functions (*the callee*) of events such as abnormal execution of the CDE triggered monitoring task, etc.

*c.* ***API-c***, used by local functional blocks to register into the corresponding repository (registry), local detected faults/errors/failures. ***API-c***, is also used by specialized incident detection mechanisms such as the monitoring tasks triggered by the CDE, to register local detected faults/errors/failures.

*d.* ***API-d***, used by local functional blocks to register locally generated alarms. The same API can be used by some local functional blocks for querying the repository (registry) about alarms received from the network. Before an alarm generator registers alarm information into the repository, it has to first request for the dissemination of the alarm to the network if necessary via the ***API-e*** of the Incident Dissemination Engine (IDE).

*e.* ***API-f*** is used by CDE clients i.e. local functional blocks to: supply at their initilisation time the fault-error-failure causality models captured during the design of a functional block. If the corresponding repository was created out of a meta-model that includes alarm information in causality relationships, then functional blocks should supply information about fault-error-failure-alarm causality models at their initialization time. During the operation lifetime of the self-managing network, human experts from the domain of fault-localization may capture more additional knowledge about causality relationships and so, would need to edit or refine causality models in the repository using special model editors that can read and store models back into the repository. This would help improve the automated fault-diagnosis and fault-removal algorithms and processes in the long term operation of the self-managing network.

*f.* ***API-g***, used by externally running monitoring tasks (outside the CDE) to supply specialized knowledge derived/obtained from monitoring, useful for fault-diagnosis.

*g.* ***NOTE:*** All the repositories can be queried for information by some functional blocks of the autonomic node. For example, the *Failure-Status* of node(s) of interest can be queried from the appropriate CDE Registry (***External Incidents Registry***) by some functional block before initiating communication with functional block(s) hosted by the target node.

*3)* **Local Incidents Registry (Cache):** For local detected Faults/Errors/Failures, which may need to be disseminated to some specified nodes where remote IDEs and CDEs store and push the info further to interested local functional blocks of their nodes. The fault-detecting (fault-detector), error-detecting (error-detector) or failure-detecting (failure-detector) entity is the one that must reason on whether dissemination is required,

depending on the network-wide impact(s) of what has been detected. There might be no need to disseminate this information to external entities. Nevertheless, all registered faults/errors/failures detected locally require an investigation. This investigation is done by the *Autonomic Node Manager* of the node, which may choose to reload modules and functional blocks or processes in collaboration with *Fault-Isolation* mechanisms of the CDE, in an attempt to achieve *self-healing (self-repair)* of the node. This Registry/Repository should be implemented based on a "*Detected Faults-Errors-Failures Meta-Model"* described in **section 4** and in **Appendix A**. Because this repository stores information/knowledge that is required by the Autonomic Node Manager (ANM) and by the CDE's core reasoning functions to perform some actions, the repository must issue an event-notification (signal) to both components upon the reception of new information/knowledge.

4) **Local and External Alarms Registry (Cache):** For alarm information, both, locally generated alarms and alarms received from the network. Functional blocks of the autonomic node may create alarms, register them with the CDE and request the IDE to disseminate locally generated Alarms to specified nodes and remote functional blocks of interest (the functional block generating the alarm specifies the recipient nodes/functional blocks of interest). This Registry/Repository should be implemented based on the "*Alarm Information Meta-Model"* described in **section 4** and in **Appendix A**. Because this repository stores information/knowledge that is required by the Autonomic Node Manager (ANM) and by the CDE's core reasoning functions to perform some actions, the repository must issue an event-notification (signal) to both components upon the reception of new information/knowledge.

5) **External Incidents Registry (Cache):** For external entities, including nodes, currently reported to be experiencing Faults/Errors/Failures and their corresponding fault-error-failure description profiles. This registry is populated by knowledge disseminated by remote IDEs and received by the local IDE and pushed into the repository. The registry is also populated by special mechanisms e.g. monitoring techniques or protocols employed/triggered by the CDE to detect failures considered to be external failures such as path, link or remote service/node-liveness failures etc. The Registry/Cache is updated (entries are deleted/cleared) by *back-to-life/recovery* messages from previously failed entities or from indications of recovery of previously faulty entities/services. Knowing the intention of some functional composition chain to communicate with affected external entities or use affected network resources, the CDE may "*raise exceptions"* based on knowledge stored in this registry. The CDE may employ some mechanisms to intercept intentions by functional blocks to communicate with the outside world or the interception can be done directly by functional blocks involved in the functional composition chain. This Registry/Repository should be implemented based on a "*Detected Faults-Errors-Failures Meta-Model"* described in **section 4** and in **Appendix**

**A**. Because this repository stores information/knowledge that is required by the Autonomic Node Manager (ANM) and by the CDE's core reasoning functions to perform some actions, the repository must issue an event-notification (signal) to both components upon the reception of new information/knowledge.

*6)* **Specialized Monitoring Info-Repository for *knowledge supplied by monitoring components*** i.e. monitoring tasks running outside the CDE. For example, the monitoring tasks that observe traffic flow and detect some patterns, abnormal events, anomalous network behavior, overload situations etc. should supply this knowledge to the CDE, to help it in reasoning about external error/failure causes (the faults) i.e. in performing fault-diagnosis/localization/isolation in collaboration with other CDEs. The error/failure causes (the description of the fault(s)) may be included in the *external-failure* (*or external-error*) notifications issued to CDE clients (local functional blocks) if known at that time, otherwise the CDE informs the clients about the fault(s) at a later time after a successful fault-localization. The CDE may trigger the gathering of more data from within the node or the network to identify the nature and severity of a failure.

*7)* **Fault-Diagnosis/Localization/Isolation functions, including network troubleshooting and debugging functions:** The CDE only executes these functions for those types of failures which may require the participation of other CDEs or which require that the CDE performs/initiates Fault-Diagnosis. All these mechanisms must be implemented inside the CDE. The core reasoning functions of the CDE use *dependability models/graphs* and *causality models/graphs* created by the functional blocks of the node during their boot-up time, and invoke appropriate fault-isolation techniques until the fault(s) is determined. As mentioned earlier, some fault-isolation and fault-removal may be implemented by the general (normal) functional blocks outside the CDE depending on the type of faults being addressed and many other factors such as design complexity, complexity of causality relationships among faults, the overhead that would be imposed on a detecting-entity in the case that the entity has to continue to provide services (i.e. is fault-tolerant), and other factors. Some fault-diagnosis mechanisms of the CDE may transform these dependency models into a so-called belief network [8] required in some fault-diagnosis. The fault-localization functions need to inform the Autonomic Node Manager (ANM) if a fault could not be localized (determined) after an attempt to localize a fault has been carried out by the functions. The ANM may then take ultimate decisions that may change the whole behaviour of the node.

*8)* **Asserted Incidents Registry (Cache):** For "*Asserted Faults/Failures*" of local entities deemed to be faulty or failing by other entities outside the node. This Registry/Repository should be implemented based on a "*Detected Faults-Errors-Failures Meta-Model*" described in section 4 and in **Appendix A**. In order to enable distributed collaborative fault-diagnosis/localization/isolation, functional blocks involved in some communication

with the outside world must be able to share views concerning faults, errors and failures that are deemed to be affecting communication goals. These views enable collaboration in hunting down the faults. Functional blocks should be able to assert that some other functional block is failing on some other node. These assertions would enable autonomic *self-examination* of the node whose functional block is deemed (reported) failing or faulty by other remote functional blocks. The self-examination would enable all the systems to collaboratively isolate the fault. Therefore, to enable this, any functional block that deems another remote functional block to be failing, needs to request its local IDE to disseminate this "assertion/view" to the CDE of the node hosting the functional block that is deemed failing. The Database client on **Figure 2** would need to request its local IDE to disseminate the alarm to the CDE of the Database server and at the same time, assert via its local IDE that the Database server is faulty or failing, possibly providing information that the designer of the client captured as possible reasons for the asserted failure. Upon receiving an asserted failure, the CDE of the Database server may trigger a self-examination algorithm that invites a CDE of another Database client to request a local Database client to send requests to the Database server while at the same time monitoring the exchange of messages between the server and this second client. Asserted faults/failures, self-examination and disseminated information on faults/errors/failures enable multiple CDEs to invite each other to perform a collaborative distributed fault-localization process. Because this repository stores information/knowledge that is required by the Autonomic Node Manager (ANM) and by the CDE's core reasoning functions to perform some actions, the repository must issue an event-notification (signal) to both components upon the reception of new information/knowledge.

*9)* **CMRepo-Repository for Fault-Error-Failure Causality Models/Graphs:** Local functional blocks create and store causality models during their initilisation according to knowledge encoded in the functional block at design time. Human experts may also edit (refine) and or add new models or more details to the models during the operation of the node.

**Note:** Repositories should have some *intelligence* implemented within, which includes the ability of a repository to reason about pushing information/knowledge to the network via the Incident Dissemination Engine (IDE) if the incident is still not cleared from the repository by recovery mechanisms, or to local components such as the Autonomic Node Manager (ANM) at some interval, especially regarding information i.e. entries pertaining to incidents (fault-activation, errors, failures, and alarms) that have not yet been cleared by recovery mechanisms.

### 5.1.3 The Incident Information Dissemination Engine (IDE)

It implements mechanisms or protocols for: *1) The dissemination of information about local detected Faults/Errors/Failures and locally generated alarms (e.g. using Gossip-based protocols/techniques, flooding, etc), 2) the dissemination of "back-to-life/recovery messages", as*

*well as the reception from the network, processing, and delivery of such messages to interested local entities, 3)reception and notifications of external faults/errors/failures to IDE clients (local functional blocks), 4) Reception of alarms from the network and delivery of the alarms to the Autonomic Node Manager of the node* or to other interested clients. A number of such mechanisms/protocols in their diversity and appropriate applicability in certain situations need to be implemented for the IDE. The selection of the appropriate mechanism to use in a particular situation is done by the IDE's core reasoning functions, and depends on time criticalness of the information to be disseminated, the need to limit the amount of dissemination related traffic in the network, and other factors. Service-failure information dissemination helps in achieving dynamic composition of alternative functions across nodes. The *back-to-life/recovery* messages are generated upon a successful fault-removal procedure when a previously reported faulty entity or service is ready to provide faultless service. Back-to-life/recovery messages/indications are generated by entities like network interface cards (NICs), a link, DNS-like type of services, as well as nodes, upon a successful recovery.

**Interfaces of the IDE:**

*a.* ***API-e***, is bi-directional and should implement two sets of functions: a set of functions that can be triggered by the IDE to disseminate information/knowledge to the registered interested local functional blocks, via say registered "callbacks". The other set of functions, whereby the IDE is the *callee*, is used by functional blocks (potential fault/error/failure detectors and alarm generators) to request the IDE if deemed necessary, to disseminate the information to the network or nodes specified by the requesting functional block. After a functional block has requested for information dissemination, it should store the information/knowledge into the corresponding repository as required for fault-diagnosis and fault-removal processes.

*b.* ***Interface with repositories (registries):*** The IDE pushes incident information (faults, errors, failures, alarms) received from the network into the appropriate repositories. Repositories meant for storing local detected incidents i.e. ***Local Incidents Registry*** *and* ***Local and External Alarms Registry*** may request the IDE to disseminate the incident information to the network for as long as it has not been cleared from the repository as an indication of recovery of the affected entity or entities.

The diagram below (**Figure 7**) shows some of the inter-working or co-operative relations between the following components: *fault-detectors, error-detectors* (all functional blocks inside the node which are able to detect errors), *failure-detectors* (all functional blocks inside the node which are able to detect service-failures and crash-failures), alarm generators, the CDE and IDEs in the network that supply knowledge to CDE via its local IDE. The diagram is a knowledge supply-oriented view of the CDE on knowledge regarding *faults, errors, failures, and alarms*. The other views such as event notifications to the CDE clients, fault-isolation, etc. are left out of the picture. Other knowledge supply-oriented views such as the supply of dependability/causality models/graphs are also left out of the picture.

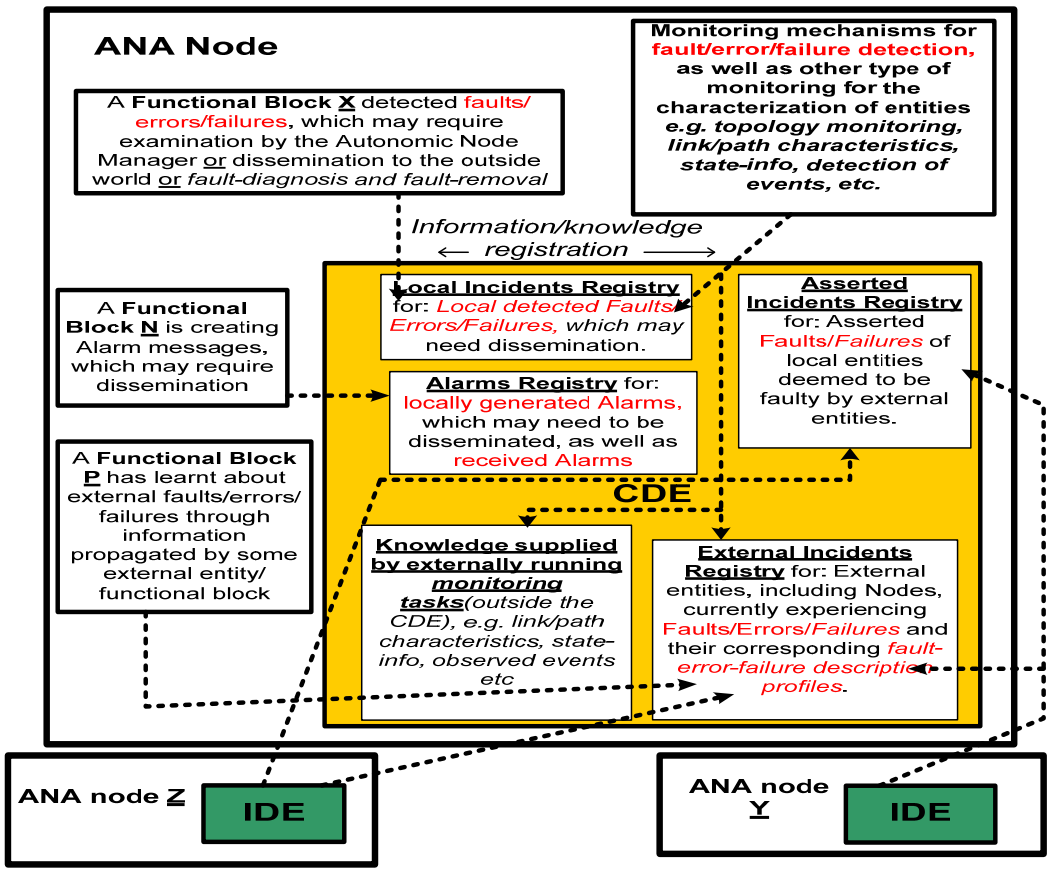


**Figure 7: Knowledge supply-oriented view of the CDE**

### 5.1.4 Dependability Models Repository (DMRepo)

This repository stores the dependability models/graphs created by the functional blocks of the autonomic node. The dependability model/graph should include context information that indicates the runtime instances involved in communication across a functional composition chain or an overlay network, including instance identifiers and possibly timers or time intervals governing the binding state between instances of functional blocks. This Repository is consulted by Fault-Diagnosis/Localization/Isolation procedures/mechanisms invoked by the core reasoning functions of the CDE. This Repository exists outside the CDE as it may need to be shared by a number of functions including those outside the CDE e.g. those running in the Autonomic Node Manager of the node. If the Dependability Meta-Model used to implement this repository includes the Faults-Errors-Failures Causality Meta-Model within itself, then this repository also stores fault-error-failure causality models/graphs.

### 5.1.5 The Autonomic Node Manager (ANM)

The ***Autonomic Node Manager (ANM)*** is an autonomous entity at runtime in relation to other functional blocks of the node. It is responsible for invoking the complex node management

functions as well as the complex network management functions requiring collaboration with other Autonomic Managers of other nodes. The ***ANM*** consists of a number of functional blocks, some of which are specific to starting and managing the components that run as "*engines*" (like the CDE and IDE), which in turn invoke specifically assigned (by design) management functions such as *Configuration management functions*, *Accounting management functions*, *Performance management functions*, *Security management functions*, and *Supervisory functions*.

Other functions of the **ANM** have already been described in section 4 and in **Appendix A**. They include a number of autonomic functions such as mechanisms to auto-detect corrupted local entities e.g. memory, crushed processes, mechanisms to perform dynamic upgrading of entities/services without service disruption, garbage collection, reloading of modules and functional blocks, replacing some corrupted entities such as modules, sending warnings to the network before rebooting the node when determined as the last resort to recovering from a Byzantine state, creating and managing spinal channels of the Spinal Cord of an Autonomic Network (SCAN) in co-operation with neighbor ANMs, as well as overseeing the health and operation of the autonomic node. Some of the operations of the ANM may need to use information about entity dependencies stored in the **Dependability Models Repository (DMRepo)** described in section 5.1.4.

# 6 CONCLUSION AND FURTHER WORK

In this Deliverable, in section 3, we presented parts of the **unified framework for implementing autonomic fault-management and failure-detection**, a framework we have developed and has been reviewed and accepted for publication by the International Journal of Network Management, John Wiley & Sons [35]. The unified framework is based on a set of nine criteria defined in the introduction section, which takes into account issues like the dynamics of a self-managing node and the dynamics of a self-managing network(s) as a whole, components and meta-models for knowledge sharing across automated processes, etc. Section 4 presents the captured and defined key components (functional blocks) as defined in the unified framework, which ought to play a role in autonomic fault-management and failure detection for self-managing networks. The architecture for ANA that is derived from the generic architecture of the unified framework follows the criteria and concepts defined in section 3. In trying to define and establish the unified framework, we presented a number of problem statements to which we provided solutions or approaches marked "**Proposition:**" along with some requirements marked "Requirement" that designers and implementers of functional blocks of an autonomic node must take into account. Clearly, the requirements capturing process must continue in order to make the framework evolve. However, the unified framework demonstrates and takes into account all the issues and concepts required to start implementing the framework, including abstractions from specific network protocol and architectural technologies, as well as abstracting from algorithms, methods and protocols for automated fault-detection, error-detection, failure-detection, fault-diagnosis, fault-removal, and information/knowledge sharing among network entities.

In section 5, we presented an architecture meant for implementation in the ANA node that is derived from the generic architecture of the unified framework. Our further work will involve addressing some of the open issues and evolvability issues discussed in [35], as well as implementing the framework in autonomic network architectures for ANA and similar projects.

# 7 REFERENCES


[1] The FCAPS management Framework: ITU-T Rec. M. 3400

[2] ITU-X.733

[3] ITU-T Rec. M. 3010

[4] R. Chaparadza: A Composition Language for Programmable Traffic Flow Monitoring in Multi-Service Self-Managing Networks. In proceedings of the 6th International Workshop on the Design of Reliable Communication Networks (DRCN 2007), La Rochelle, France.

[5] M. Steinder, A. S. Sethi: A survey of fault localization techniques in computer networks: Copyright Elsevier: Published in the Journal – Science of Computer Programming 53 (2004) 165-194.

[6] R. Chaparadza: Improving the automation of network management and troubleshooting tasks by applying On-Demand Monitoring of protocol(s) specific traffic in multi-service networks. 5th IEEE International Workshop on IP Operations & Management, IPOM 2005, in the local proceedings. Barcelona, Spain, October 2005.

[7] L.Kant, A.S. Sethi and M. Steinder: Fault localization and self-healing mechanisms for FCS networks: Proc. 23rd Army Science Conference, Orlando, FL (Dec. 2002).

[8] M. Steinder, A. S. Sethi: Probabilistic Fault Localization in Communication Systems Using Belief Networks: In IEEE/ACM Transactions on Networking, Vol 12, No. 5, October 2004.

[9] C. Wang and M. Schwartz: Fault Detection with Multiple Observers: In IEEE/ACM Transactions on Networking, Vol. 1, No.1, February 1993.

[10] A. Avizienis, J. C. Laprie, B. Randell, C. Landwehr: Basic Concepts and Taxonomy of Dependable and Secure Computing: In IEEE Transactions on Dependable and Secure Computing, Vol. 1, No. 1, January-March 2004.

[11] O. Laouamri and C. Aktouf: Towards a Complete SNMP-Based Supervision of System-on-Chips: In Journal of Network and Systems Management, volume 13, Number 4, December 2005, published by Springer.

[12] R. Chaparadza, H. Coskun, I. Schiferdecker: Addressing some challenges in autonomic monitoring in self-managing networks. IEEE International Conference on Networks and IEEE Malaysia International Conference on Communications and (MICC & ICON 2005), Kuala Lumpur, Malaysia.

[13] A. Abbas: The Autonomic Computing Report – Characteristics of Self Managing IT Systems. Grid Technology Partners, 2003.

[14] M. Smirnov: Autonomic Communication, Research Agenda for a New Communication Paradigm. White paper, Fraunhofer FOKUS, 2003.

[15] Self-Managing Networks: Building the Infrastructure for Utility Computing AT&T White paper.

[16] R. Mortier, E. Kiciman: Autonomic Network Management: Some Pragmatic Considerations: SIGCOMM'06 Workshops, September 11-15, 2006, Pisa, Italy.

[17] R. Sterritt, D. Gunning, A Meban, P. Henning: Exploring Autonomic Options in an Unified Fault Management Architecture through Reflex Reactions via Pulse Monitoring: Proceedings of the 11th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems (ECBS'04).

[18] The Autonomic Nervous System of a human:

http://users.rcn.com/jkimball.ma.ultranet/BiologyPages/P/PNS.html#autonomic

[19] The Spinal Cord of a human:

http://users.rcn.com/jkimball.ma.ultranet/BiologyPages/C/CNS.html#SpinalCord

[20] The Model Driven Architecture (MDA) from OMG: http://www.omg.org/mda/specs.htm

[21] The MOF language: Meta Object Facility (MOF) 2.0 Core Specification: http://www.omg.org/mda/specs.htm#MOF

[22] The XML Metadata Interchange ( XMI) Language: http://www.omg.org/mda/specs.htm#XMI

[23] Y. Tang, E. Al-shaer, R. Boutaba: Active Integrated Fault Localization in Communication Networks: in IEEE/IFIP Integrated Management (IM'2005), May 2005.

[24] R. R. Kompella, J. Yates, A. Greenberg, A.C. Snoeren: Detection and Localization of Network Black Holes: In Proceedings of Infocom 2007.

[25] N. Ramanathan, K. Chang, R. Kapur, L. Girod, E. Kohler, D. Estrin: Sympathy for the Sensor Network Debugger: SenSys'05, November 2–4, 2005, San Diego, California, USA.

[26] A. Aghasaryan, C. Dousson, E. Fabre, A. Osmani, Y. Pencole: Modeling Fault Propagation in Telecommunications Networks for Diagnosis Purposes: proceedings of WTC'02 (World Telecommunications Congress), Paris, France, September, 2002.

[27] A. Saple, L. Yilmaz: Agent-based Simulation Study of Behavioral Anticipation: Anticipatory Fault Management in Computer Networks: In Proceedings of the 44th annual Southeast regional conference, ACM Southeast Regional Conference.

[28] J. Dunagan, N. J. A. Harvey, M. B. Jones, D. Kostic: FUSE: Lightweight Guaranteed Distributed Failure Notification: In Proceedings of 6th Symposium on Operating Systems Design and Implementation (OSDI 2004), 2004.

[29] John C. Strassner etal: FOCALE – A Novel Autonomic Networking Architecture: In proceedings of the Latin American Autonomic Computing Symposium (LAACS), Campo Grande, MS; Brazil 2006.

[30] R. Braden, T. Faber, M. Handley: From Protocol Stack to Protocol Heap – Role Based Architecture: In ACM SIGCOMM Computer Communications Review, Volume 33, Number 1: January 2003.

[31] C. Tschudin: Flexible Protocol Stacks: In ACM SIGCOMM Computer Communication Review, Volume 21, Issue 4 (September 1991).

[32] A. Hanemann, M. Sailer, D. Schmitz: Assured Service Quality by Improved Fault Management: ICSOC'04, November 15–19, 2004, New York, New York, USA.

[33] European IST FP6 ANA(Autonomic Network Architecture) Project: http://www.ana-project.org/

[34] J. P. Vasseur, M. Pickavet, P. Demeester: Network Recovery, Book published by Morgan Kaufmann, Copyright 2004 by Elsevier.

[35] R. Chaparadza: A Unified Framework for Implementing Autonomic Fault-Management and Failure-Detection for Self-Managing Networks: reviewed and accepted for publication by the International Journal of Network Management, John Wiley & Sons, (to be published in 2008).

[36] ANA Deliverable 3.2 (M12): First draft on the resilience and security framework

[37] ANA Deliverable 3.1 (M12): ANA Monitoring Framework Draft

# APPENDIX A

## Overview of mechanisms for fault, error and failure detection

The following is a brief overview of mechanisms for fault, error and failure detection. We give the categories of the mechanisms or methods of detecting faults, service-failures and component or device failures.

- Specialized monitoring protocols or techniques i.e. *active monitoring or active-probing* methods [23][32]. Among these methods we find the *Heartbeat detection* method found in protocols like ATM, SDH and BFD that uses *keep-alive messages* exchanged at a *prescribed interval of time* (by using timers). If a *heartbeat is missed*, the path, link, device or node is declared as failed and a reaction action such as a switchover is performed.
- A *Functional Block(s)* that uses a service provided by some other functional-block through the *service access point(s)* of the provider functional block is considered a *potential failure-detector* of the service(s) provided via the service access point.
- A peer *Functional Block* e.g. a protocol entity can detect that a peer(s) is failing to provide correct service.
- Passive monitoring techniques [23]. In this category, a specially instrumented passive probe(s) listens for traffic or messages from a system/component under observation and implements some analysis functions that reason about the timing of observed messages or traffic and the correct semantics of the observed messages or traffic in order to declare that the system or component under observation is faulty, failing or malfunctioning.
- The failed or failing "System, Component, Functional Block e.g. a Protocol" may have its own internal mechanisms to detect and communicate its own failures to provide a requested service in addition to mechanisms to detect internal and external faults.
- Error messaging methods to enable entities to learn about the presence or activation of a fault somewhere, or to indirectly learn about a faulty entity.

## The required Meta-Models (Information models)

### The "Dependability Meta-Model"

It is an "information model" that describes concepts and relationships that enable the creation of information and relationships about entities (functional blocks, services, functions, components, nodes etc). For *autonomic fault-management*, we propose that the dependability model be created automatically and co-operatively by the functional blocks of nodes involved in some communication context and the model must include run-time context information. A dependability model has a macro-view and micro-view, which is a refinement of the macro-

view, to include a view of the relationships within particular model components e.g. possibly complex relationships within services and functions within a peer-relationship [7]. The repository that stores information/knowledge about dependencies between functional blocks, functions, services, and nodes should be populated during the boot-up time of functional blocks of the autonomic node as well as during the operation of functional blocks as context information e.g. bindings changes.

**Figure 3** below shows a dependability model/graph of services provided through interfaces of functional blocks and the functions implemented by the functional blocks whose interactions with peer functional blocks provide the services. The vertical dotted arrows show dependency relationships. The curved dotted bi-directional arrows reflect the concept of a service and the functions that implement it. The difference between the model we present here and the layered model presented in [5] is that, here we are looking at functional blocks and the notion of functional composition chain with no restrictions on the OSI-layering approach. Clean slate approach type of projects like ANA [33] (also refer to [30] and [31]) are calling for flexibility in the way network functions are designed, to allow network functions to be implemented without following the notion of the traditional OSI layer functions but rather the functions are designed as functional modules or units that can be composed or bundled on-demand at run-time into a chain that can flexibly be changed or re-composed. This means that each functional unit or simply a functional block needs to be designed in such a way that its functional capabilities can be queried by a composing entity in order to wire together functional blocks into a stack that fulfils some communication purpose. This also means a functional block may actually implement a number of functions that allow the functional block to have more than one peer-functional block in the network-wide communication functional block chains of multiple nodes (see functional block 1 on **Figure 3** below). For instance, a functional block may have multiple peers depending on the functions it implements and, not the same number of functional blocks may be required in two communicating end systems i.e. the numbering N and M do not have to be the equal. In a layered model N, X and M would be equal and implementing functions of the same layer. ANA [33] is one of such projects that talk about functional blocks and not layers in the strict sense. Whether it's a strictly layered model as in [5] or it is a kind of model as the one presented here, still, a dependability meta-model needs to be captured and designed.

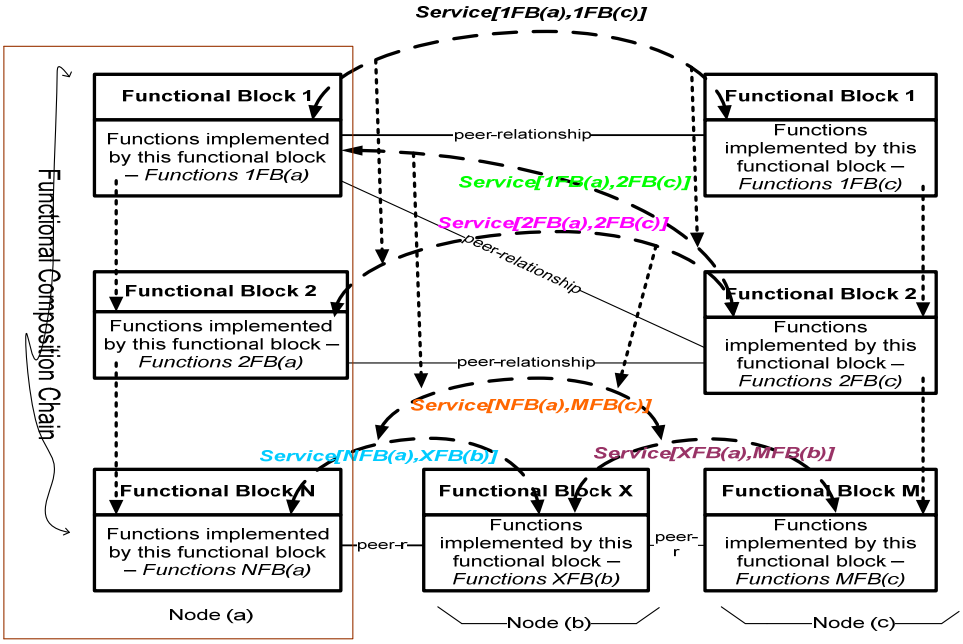


**Figure 3: Dependability model/graph of services and functions**

*Requirement:* Every functional block *must* create a micro-view of the services and functions that it implements. This micro-view is part of the *dependability model/graph* and consists of micro-models whereby a micro-model of a *Service [XFB(a), XFB(c)]* or *Function XFB(a)* defines which failure mode occurs in *Service[XFB(a), XFB(c)]* or *Function XFB(a)* when a particular *failure mode* (see earlier descriptions on failure modes or refer to [7][10]) occurs in a service or function on which *Service [XFB(a), XFB(c)]* or *Function XFB(a)* depends. Refer to **Figure 3** for the picture on dependency relationships. This means, the implementation of every functional block must know the *Dependability Meta-Model* in order to instantiate some concepts and relationships, and store/register the information/knowledge into the corresponding repository. If the corresponding repository is implemented as a MOF-based repository rather than say a relational database, then, any functional block that accesses it should be compiled and linked with the IDL skeletons of the repository's interface(s) or should use some specially compiled libraries to talk with the repository. However, there are situations whereby the functional blocks of an autonomic node are composed by some central logic we may call a Functional Composition Logic (FCL) that composes functional blocks on demand according to their capabilities. This is the case in the ANA [33] project, where a so called Functional Composition Framework composes functional blocks for the need of provisioning some service of some characteristics. In such cases, the Functional Composition Logic (FCL) must work co-operatively with the composed functional blocks to create and store the dependability modes/graphs into the appropriate repository. Mechanisms that allow nodes to co-operatively share knowledge about dependencies spanning multiple functional blocks in different nodes are required.

## The "Faults-Errors-Failures Causality Meta-Model"

It is an "information model" that describes concepts and relationships that enable the creation of knowledge about what type of fault(s) causes what type of error(s), what type of error(s) causes what type of a failure(s), what type of failures cause what type of faults, and so on and so on, as a

chain. The causality relations must include the components and points/interfaces in the network that affect each other, to complement relationships already captured by the Dependability Meta-Model. The repository that stores this information should be populated during the boot-up time of functional blocks of the autonomic node. *Requirement:* Every functional block of the node *should* create a *causality model/graph* in the corresponding repository by creating information about: *fault-error-failure causality relationships* captured during the design and implementation of the functional block. The functional block creates the information during the boot-up phase. This means, the implementation of the functional block must know the *Faults-Errors-Failures Causality Meta-Model* in order to instantiate some concepts and relationships, and store/register the information/knowledge into the corresponding repository. If the corresponding repository is implemented as a MOF-based repository rather than say a relational database, then, any functional block that accesses it should be compiled and linked with the IDL skeletons of the repository's interface(s) or should use some specially compiled libraries to talk with the repository. However, causality graphs/models can also be created by human experts using an appropriate editing tool that then writes/stores them into the repository. The human experts may also need to edit some graphs/models automatically produced and stored in the repository by the functional blocks of the node. This is because, not all fault-error-failure causality model relationships can be captured at the design phase.

## The "Detected Faults-Errors-Failures Meta-Model"

It is an "information model" that describes concepts and relationships that enable the creation of information about detected faults, errors, failures and other information useful for fault-isolation and fault-removal. The repository that stores this information should be populated dynamically as incidents occur. By design, a functional block of an autonomic node must be designed in such a way that when it detects an internal error or detects an external error (failure), directly or through error-propagation or error-reporting by other functional blocks (local or remote), the functional block must self-describe these incidents, following this meta-model, and push the knowledge into the repository. Therefore, all potential fault-detectors, error-detectors and failure-detectors inside an autonomic node must behave as described above. The functional block(s) solely dedicated to exercising issues like fault-removal i.e. the Autonomic Node Manager (ANM) of the node accesses the corresponding repository in order to read the information models and invoke algorithms for fault-removal.

*Requirement:* A functional block that detects faults/errors/failures and registers this knowledge into the corresponding repository *must* include the following information (as described by appropriate concepts of the meta-model) in its knowledge description:

- the *identifier* of the detecting entity (the functional block) i.e. process or thread identifiers, etc.
- the *identifier* and *interface description/identifier* of the *provider functional block* whose service was being requested when a error/failure detection occurred. In ANA, this should include the *identifiers* of such concepts as the Information Dispatch Points (IDPs), the Information Channels (ICs) involved, etc.
- what has been detected and the corresponding *timestamp*.
- service-description and the identifier if known, and the *failure mode* if a failure has been detected,
- the critical-ness/severity of the problem.

- the viewpoint, whether it's an internal fault/error/failure or external.
- Any other information helpful in fault-diagnosis and fault-removal.

This means, the implementation of the functional block must know the *Detected Faults-Errors-Failures Meta-Model* in order to instantiate some concepts and relationships, and store/register the information/knowledge into the corresponding repository. If the corresponding repository is implemented as a MOF-based repository rather than say a relational database, then, any functional block that accesses it should be compiled and linked with the IDL skeletons of the repository's interface(s) or should use some specially compiled libraries to talk with the repository.

The "*Detected Faults-Errors-Failures Meta-Model*" and the "*Faults-Errors-Failures Causality Meta-Model*" capture common concepts, the difference being that the "*Detected Faults-Errors-Failures Meta-Model*" adds additional concepts that are associated with describing points or interfaces of incident occurrence (detection) including identifiers of run-time entities detecting the incident or on which the incident has been detected, whereas the "*Faults-Errors-Failures Causality Meta-Model*" focuses mainly on causality relationships even among distributed entities, mainly captured during design of functional blocks. Therefore, it is important that the vocabulary used to describe faults, errors and failures during the instantiation of either of the two meta-models is common and standardized to enable the development of algorithms such as fault-diagnosis or fault-removal that operate on instances of these two meta-models i.e. models.

## The "Alarm Information Description Meta-Model"

It is an "information model" that describes concepts and relationships that enable the creation of alarm information descriptions. The repository that stores models of this meta-model should be populated by the node's local functional blocks that generate alarms during the operation of the node. Alarms received from the network are also stored in this repository. The Autonomic Node Manager of the node may need to read information about locally generated alarms and alarms received from the network in order to trigger self-adapting functions, generate or enforce new policies that the functional blocks of the node should adhere to. Fault-diagnosis functions also read information from the alarms repository in order to perform alarm correlation and fault-localization. Requirement: When designing functional blocks the designer should ensure that a functional block that generates an alarm(s) includes in the generated alarms, information that will help with determining the cause of the potentially abnormal situation e.g. the probable cause, including information identifying the alarm generating entity, and other information related to side effects as pointed out in [2]. All such information should first be captured via the definition of the alarm information description meta-model, way before functional blocks of the autonomic node are designed. **Proposition:** In order to design this meta-model, a number of concepts related to alarm information can be adopted from the TMN framework [2], provided that the adopted concepts are suitable for self-managing networks. Because alarm information should include probable causes, the Alarm Information Description Meta-Model could be merged with the Faults-Errors-Failures Causality Meta-model into a Faults-Errors-Failures and Alarms Causality Meta-Model in order to capture the causalities within the same information modeling framework. If the corresponding repository is implemented as a MOF-based repository rather than say a relational database, then, any functional block that accesses it should be compiled and

linked with the IDL skeletons of the repository's interface(s) or should use some specially compiled libraries to talk with the repository.

## The "Meta-Model of Knowledge Supplied By Monitoring Components"

It is an "information model" that describes concepts and relationships that enable the creation of information which describes specialized monitoring information required in Fault-Diagnosis e.g. *path-characteristics such as delay, round-trip-time, load, link utilization, those monitored events that can not be classified as alarms, such as events detected regarding status changes or traffic flow, global network state information, topology-description info, topology changes, etc.* If the corresponding repository is implemented as a MOF-based repository rather than say a relational database, then, any functional block that accesses it should be compiled and linked with the IDL skeletons of the repository's interface(s) or should use some specially compiled libraries to talk with the repository.

## The use of knowledge/information shared through repositories in autonomic fault-management and failure-detection processes

In automated distributed collaborative fault-diagnosis, one of the key assets required are *dependability models/graphs* that reflect information and relationships about entities (functional blocks, services, functions, nodes, etc). Also, knowledge about what type of fault(s) causes what type of error(s) and what type of error(s) causes what type of a failure(s), what type of failure(s) cause what type of fault(s), and so on and so on, captured through *causality models/graphs* is required in the automated fault-diagnosis and fault-removal processes. For *automated collaborative fault-diagnosis and fault-removal, one of the requirements for autonomic fault-management*, dependability models that include run-time context information of functional blocks and the faults-errors-failures causality models form knowledge that must be created automatically by functional blocks of the autonomic nodes during their boot-up time. The functional blocks of an autonomic node must push this knowledge into the node's repositories in order to share the knowledge with functional blocks that can operate on it such as an Autonomic Node Manager, Fault-Diagnosis functional blocks, etc. This means either a functional block of an autonomic node must be designed in such a way that it knows the functional blocks it depends on or a specially designed higher level central component (logic) knows dependencies between functional blocks of the node, and can use say the IDL interface of an appropriate repository in order to store information required during the boot-up time, such as functional block dependencies as well as causality relationships among faults, errors, and failures.