Aalto-yliopisto
Sähkötekniikan
korkeakoulu

César Marco Rubio

Study of the Applicability of Model-Driven Methodologies for the Design of Autonomic Behaviours

Master's Thesis submitted for examination for the degree of Master of Science in Technology.

Espoo, May 29, 2011

Supervisor: Professor Jörg Ott
Instructor: M.Sc. András Zahemszky

AALTO UNIVERSITY
SCHOOL OF ELECTRICAL ENGINEERING

ABSTRACT OF THE
MASTER'S THESIS

Author:	César Marco Rubio
Name of the thesis:	Study of the Applicability of Model-Driven Metodologies for the Design of Autonomic Behaviours
Date:	May 29, 2011 **Number of pages:** 100+12
Department:	Department of Communications and Networking
Professorship:	S-38
Supervisor:	Prof. Jörg Ott
Instructor:	M.Sc. András Zahemszky

The complexity and costs needed for network management are currently very high, thus the networking industry is calling for a change in the network management area that would reduce these two aspects. The solution is based on moving some of the management tasks that involve human intervention into the network itself, creating autonomic networks. EFIPSANS is a project that aims at exploiting and extending the features of IPv6 and related protocols to enable the realization of IPv6-based autonomic and self-managing networks and services. The self-management features that EFIPSANS has developed are known as Autonomic Behaviours, which are realized by control-loops within a system e.g. a router or within the overall network as a system. The architecture developed within the project for designing and engineering Autonomic Behaviours is Generic Autonomic Network Architecture (GANA).

The goal of this Master's Thesis is to contribute to the development of a Model-Driven Methodology and an associated Tool-Chain that can be applied for the design, simulation, verification and validation of Autonomic Behaviours.

In this thesis, first, we give an overview on the EFIPSANS project, focusing on the different Autonomic Behaviours, as well as on the GANA reference model. Then, we discuss the identified Model-Driven Methodology and the implementation details of the Tool-Chain, which is orchestrated by several tools of different natures. Then, we show a step-by-step case study using the developed Tool-Chain. Finally, we discuss the benefits and the limitations of the implemented Tool-Chain.

Keywords: Autonomic Behaviours, GANA, Tool-Chain, Model-Driven Methodology

AALTO-YLIOPISTO
SÄHKÖTEKNIIKAN KORKEAKOULU

DIPLOMITYÖN
TIIVISTELMÄ

Tekijä	César Marco Rubio
Työn nimi:	Study of the Applicability of Model-Driven Metodologies for the Design of Autonomic Behaviours
Päivämäärä:	29.5.2011 **Sivuja:** 100+12
Tiedekunta:	Tietoliikenne- ja tietoverkkotekniikan laitos
Professuuri:	S-38
Työn valvoja:	Prof. Jörg Ott
Työn ohjaaja:	M.Sc. András Zahemszky

Verkonhallinta on tällä hetkellä monimutkaista ja vaatii korkeat kustannukset. Näin ollen verkkoteollisuus tarvitsee verkonhallinnan alueelle muutoksen, joka vähentää näitä kahta aspektia. Ratkaisu perustuu siihen, että osa hallintatoiminnoista, jotka tarvitsevat ihmisen puuttumista, siirretään itse verkkoon, luoden atonomisia verkkoja. EFIPSANS on projekti, joka tähtää IPv6:n ja siihen liittyvien protokollien ominaisuuksien hyödyntämiseen ja laajentamiseen, jotta IPv6-pohjaisten autonomisten ja itseohjautuvien verkkojen ja palvelujen realisointi olisi mahdollista. EFIPSANS:n kehittämät itseohjautuvuuden ominaisuudet tunnetaan Automisina Käyttäytymisinä, jotka toteutetaan järjestelmässä, kuten reitittemessä, tai kokonaisen verkon muodostamassa järjestelmässä, kontrollointi silmukoiden avulla. Projektissa suunniteltu arkkitehtuuri Autonomisten Käyttäytymisten suunnitteluun ja toteuttamiseen kutsutaan Yleiseksi Autonomiseksi Verkkoarkkitehtuuriksi (GANA).

Diplomityön tavoitteena on edistää Malliperusteisen Metodologian ja siihen liittyvän Työkaluketjun kehittämistä, jota voidaan soveltaa Autonomisten Käyttäytymisten suunnitteluun, simulointiin, todentamiseen ja hyväksymiseen.

Tämä diplomityö alkaa johdannolla EFIPSANS projektiin, pääpainonaan eri automiset käyttäytymiset, kuin myös GANA referenssimalli. Sen jälkeen käsittelemme Malliperusteista Metodologiaa sekä useista erityyppisistä työkaluista koostuvan Työkaluketjun toteutuksen yksityiskohtia. Lopulta käsittelemme toteutetun Työkaluketjun etuja ja rajoituksia.

Avainsanat: Autonomisten Käyttäytymisten, GANA, Työkaluketjun, Malliperusteista Metodologiaa

Acknowledgments

This Master's thesis has been conducted in the NomadicLab research laboratory at Ericsson Research Finland, as part of the EFIPSANS project collaboration. I wish to thank Tony Jokikyyny for providing me with the great opportunity to carry out my thesis at his research section.

I wish also to thank my thesis instructor, András Zahemszky, for all his valuable comments and efforts. In addition, I would like to thank all my friends at Ericsson Finland; especially to Gonzallo Camarillo, whose correct words made me select this Master's degree.

I would also like to thank my thesis supervisor Jörg Ott for his flexibility and valuable feedback.

I owe a special thank to Arun Prakash, who has helped me a lot along the process of this thesis.

I wish to thank my parents and brother for having always provided me the correct support for reaching my goals.

Finally, I want to provide my deepest gratitude to Nadine Wollenick for supporting me along the studies and being such a great inspiration.

Helsinki, May 29, 2011

César Marco Rubio

Contents

Abbreviations

3GPP	3rd Generation Partnership Project
AB	Autonomic Behaviour
ABs	Autonomic Behaviour Specification
BGPv4	Border Gateway Protocol version 4
CL	Control Loop
CM	Configuration Management
COM	Component Object Model
DE	Decision Element (synonymous with DME)
DHCPv6	Dynamic Host Configuration Protocol for IPv6
DME	Decision-Making Element (synonymous with DE)
EFIPSANS	Exposing the Features in IP version Six protocols that can be exploited/extended for the purposes of designing/building Autonomic Networks and Services
EMF	Eclipse Modelling Framework
FCAPS	Fault, Configuration, Accounting, Performance and Security
FIB	Forwarding Information Base
FM	Fault Management
FSM	Finite State Machine
GANA	Generic Autonomic Network Architecture
GME	Generic Modelling Environment
GUI	Graphical User Interface
HCLs	Hierachical Control Loops
ICMPv6	Internet Control Message Protocol version 6
IP	Internet Protocol
IPv6	Internet Protocol version 6
ISP	Internet Service Provider
MANET	Mobile Ad hoc NETworks

MARSIAN	Management, Auto-configuration, Resilience and Survivability In Mobile Ad hoc Networks
MDE	Model-Driven Engineering
ME	Managed Entity
MOF	Meta-Object Facility
MS	Microsoft
MSC	Message Sequence Charts
NMS	Network Management Systems
OMG	Object Management Group
ONIX	Overlay Network for Information eXchange
OPEX	Operational Expenditure
OSPFv3	Open Shortest Path First version 3
P2P	Peer-To-Peer
QoS	Quality of Service
RIB	Routing Information Base
RM	Routing Management
SDL	Specification and Description Language
SNMP	Simple Network Management Protocol
SOA	Service-Oriented Architecture
SOHO	Small Office/Home Office
TCP	Transmission Control Protocol
TNM	Telecommunication Network Management
UDP	User Datagram Protocol
UML	Universal Modelling Language
UPPAAL	Uppsala University / Aalborg University
VBA	Visual Basic for Applications
XMI	XML Metadata Interchange
XML	Extensible Markup Language

List of Figures

List of Tables

Chapter 1

Introduction

Telecommunications Network Management (TNM) [1] is the technology within the networking industry that aims to administrate, monitor and control computer networks. This concept groups the abstract and physical definitions, the purpose of which is to carry out overall management of the communication networks. It includes all those activities, procedures, tools and devices that intend to achieve any of the tasks associated with the network management concept form part of the Telecommunications Network Management technology.

The area of Network Management has been in continuous and simultaneous development with the development and expansion of communication networks. All the technologies associated with TNM are thus becoming increasingly important to the network providers and administrators as the architectural complexity of communication networks increases.

In order to better understand the relevance that the management of the networks has for administrators and operators in today's networks, it is worthwhile to look into the history and evolution of the networks and the different techniques applied when managing them. As it is stated in J. Richard Burke's book "*Network Management Concepts and Practice: A Hands-on Approach*" [2], in the early stages of communication networks, network administrators were able to manage their networks and associated devices in an individual manner, in other words, by individually checking each of the devices, one after the other. This way of working did not scale and the larger the networks became the higher investment in personnel for managing the network was required. As

a consequence, higher expenses in salaries and other employment related costs were incurred for the owner of the network.

Network Management Systems (NMS) were designed and developed for solving the management problems that network operators faced during these early years. NMSs are centralized systems developed to manage, control, monitor and operate communication networks and to drastically reduce the expenses associated with the administration of different network technologies. The ITU-T (International Telecommunication Union - Telecommunications Sector) outlined the TNM and NMSs in the recommendation M.3010 "Principles for a Telecommunications management network" [1]. TNM and NMSs were further defined by the 3rd Generation Partnership Project (3GPP) in the specification 32.101 "Telecommunication management; Principles and high level requirements" [3]. In these technical documents the network management framework, which is based on the FCAPS concept, is specified. FCAPS stands for Fault Management, Configuration Management, Accounting Management, Performance Management and Security Management.

The NMSs can range from very simple systems, such as a single application installed in a computer, to very complex architectures built with several servers and many applications running on them. However, all of them have the same purpose: to provide the system administrator with the proper tools for managing the networks, increasing the efficiency of the administration tasks. Network administrators manage their networks from the centralized entity of the NMS, where an application communicating in client-server style with the rest of the devices is running. Network administrators use this application for managing their networks.

The devices comprising today's network architectures are designed to follow the FCAPS concept. The network elements will feed the centralized system with status information and events. Later, the centralized system will analyze and evaluate the received information and in most of the cases display the results to the administrator. In the centralized systems the necessary administration decisions are made, either through the administrators or via automatic procedures. These decisions will be pushed into the devices, where some procedures/processes will be triggered driving their behaviour to a certain goal. Thus, this technology totally relies on the centralized control of the network.

On the other hand, this kind of centralized control of the network has some drawbacks. As with any other centralized system, there is one single point of failure for the whole administration of the network, i.e. if the central system is down, the operator will not be able to properly control the network. Requiring these systems for the administrative processes increases considerably the operational expenditure (OPEX) of the network operators. In addition, network operators consider the current solution for managing the networks too complex [6].

However, there is an alternative approach to network management that has become more and more relevance in the research and industrial sectors over the past years [4] [5]. It is based on decentralizing the management of the network, moving some of the intelligence to the devices themselves. The final purpose of such an approach is to create autonomic networks, i.e., self-managing networks.

This work is based on one of the projects covering the topic of autonomic networks: EFIPSANS. The acronym EFIPSANS stands for "*Exposing the Features in IP version Six protocols that can be exploited/extended for the purposes of designing/building Autonomic Networks and Services*". At it is stated in the web page of the project [5]:

> *"The EFIPSANS project aims at exposing the features in IP version six protocols that can be exploited or extended for the purposes of designing or building autonomic networks and services."*

In Chapter 2.1 we provide an extended introduction to the EFIPSANS project, where the ideas and technologies used for creating the Autonomic (Self-Manageable) Networks are described.

EFIPSANS is an EU project that aims to expand/extend the capabilities of IPv6 and related protocols in order to develop the future of the network management area. The intention of the project is to develop techniques for moving some of the management tasks to the network itself, evolving them into Autonomic and Self-Managed Networks. The self-manageable processes that EFIPSANS is developing are known as Autonomic Behaviours. These Autonomic Behaviours take care of every aspect of the network, e.g. router configurations, routing functions, QoS, mobility, protocol configurations, global network behaviour, etc.

The Autonomic Behaviours aim to reach a desirable behaviour for the whole network by managing the mentioned networking features.

The architecture for carrying out these Autonomic Behaviours is termed GANA (Generic Autonomic Network Architecture). GANA is based on 4 hierarchical control-loops running at different levels of the network structure: protocol, function, node and network levels. These hierarchical control-loops are fed with feedback information from different levels. This information is then internally analyzed according to possible characteristics of the network's functionality. Later, the processed information is passed to a Decision Element (DE), which takes a decision to alter some of the characteristics of their underlying managed systems, leading to the required behaviour. These decisions are taken based on policies set by the network administrator.

1.1 Objectives and Scope of the Thesis

The EFIPSANS project acknowledges the complexity involved in the design of the Autonomic Behaviours and their control-loops. Thus, one of the goals of the EFIPSANS project was to develop a Model-Driven Methodology and an associated Tool-Chain that can be applied into GANA's architecture development process in order to help in the design, simulation, verification, validation and testing of the Autonomic Behaviours. The work carried out during this thesis has been done in collaboration with the tasks involved on this goal.

In the scope of this thesis, we have contributed in the implementation and testing of the applications forming the Tool-Chain. In addition, some of the processes involved in the methodology have been evaluated and some enhancement ideas discussed.

The final objective of this Master Thesis is to study and evaluate the applicability of Model-Driven Methodologies and Tool-Chain techniques in the development of Autonomic Behaviours. For such purpose, we will design an Autonomic Behaviour by using the proposed Model-Driven Methodology and the developed Tool-Chain. This Autonomic Behaviour will perform some self-management tasks in a defined networking scenario. The final results obtained from applying these techniques in the design of the Autonomic Behaviour will be analyzed and discussed.

1.2 Structure of the Thesis

This thesis is structured in three main blocks: the study and analysis of the EFIPSANS project, the deep description of the Model-Driven methodology and Tool-Chain involved, and the creation of a case study using these techniques. In Chapter 2 the EFIPSANS project is described from the networking point of view. In Chapters 3 and 4 the Model-Driven techniques and the designed Tool-Chain are analyzed and studied. In Chapter 5 the case study is evaluated. Finally, I provide the conclusions of the thesis project in Chapter 6.

Chapter 2

Background

In this chapter the background information of the Thesis topic is reported. First, we describe the EFIPSANS project as such, pointing out the motivations that had triggered it and the goals the project itself has aimed to achieve. Later, a deeper description of the theory investigated in the project is done by explaining the GANA architecture and the Autonomic Behaviours.

2.1 EFIPSANS as a Project

EFIPSANS (Exposing the Features in IP version Six protocols that can be exploited/extended for the purposes of designing/building Autonomic Networks and Services) [5] is the name of an EU funded project aiming to design and develop autonomic networks by exploiting or extending the features of IPv6 and related protocols. As it was introduced in Chapter 1, the idea of developing techniques to move some of the management tasks of current networks to the network itself is becoming increasingly important in the industry. The ultimate purpose is of the EFIPSANS project is creating self-manageable networks. Developing such techniques would enable the networks and their individual devices to be context-aware and self-adaptive in those administration challenges that nowadays are managed by the NMSs. As described in [7], the autonomic networks conceptualized by EFIPSANS should achieve *robust, scalable, resilient, self-healing, self-configuring, self-recovering Internet dimensions in the context of Future Internet*.

EFIPSANS is a multi-partner project involving 15 different stakeholders from 11 different countries [5]. The partners of the project come from the academic community, the network vendor industry, the network provider industry and the research industry.

2.1.1 Motivation

The fast evolution of the networks has resulted in a growth of the associated complexity. It is a widely-acknowledged fact that network operators are facing problems due to the growing complexities of their networks, which in turn leads to an increase in administration activity costs. On the other hand, cost reduction is an essential key for maintaining or enhancing market position [6]. Furthermore, the management burden that the networks require is limiting the capacity of the operators to invest in the development of new technologies or services.

These kinds of problems were the catalyst for the idea of moving some of the management tasks to the network itself, this would generate a reduction of administration expenses and usage of human intervention in the management tasks. At the same time, specialist and network researchers were also considering the architectural design of the Future Internet. In order to merge both necessities two different approaches were discussed: an evolutionary approach, where today's networks are evolved to fulfil the requirements; and the revolutionary approach, where the design of the future networks is done from a clean-slate [7]. The latter approach has given rise to such well known projects as 4D [8], CONMan [9] or ANA [4], while the evolutionary approach enabled such projects as FOCALE [10].

However, after reviewing many of the existing projects, the EFIPSANS project concluded that none of these approaches entirely fulfilled the generic requirements for designing the architecture of a self-managed network from an evolvable point of view. In order to overcome this situation, the EFIPSANS project proposed to create a holistic Generic Autonomic Network Architecture (GANA) to be used as a reference model in the development of autonomic networks. The GANA architecture is further discussed in Chapter 2.2.

In order to achieve an evolvable approach for the GANA architecture, EFIPSANS evaluated that the key player providing the potential characteristics for

accomplishing the desired self-management networks was the Internet Protocol version 6 (IPv6). The IPv6 protocol provides a considerable enhancement of communication possibilities in comparison to its processor the IPv4, such as auto-configuration, neighbour discovery or flexible protocol extensibility [11]. EFIPSANS considered that the possibilities offered by IPv6, together with its growing acceptance, would make the IPv6 protocol, potential extensions and related protocols of the enablers of the future autonomic networks.

2.1.2 Goals, Objectives and Outcomes

The EFIPSANS project aims to design and develop the holistic reference model to be used for the creation of autonomic networks. This reference model will include the specification of the network players, their interfaces and interactions, and the behaviours that would drive self-management activities. In addition, it is aimed to specify the GANA characteristics for helping software developers to better understand the architecture of this approach. This is achieved by the development of the GANA Meta-Model and associated Advanced Methodologies [12].

Furthermore, as it is stated in the kick-off meeting presentation [12], the EFIPSANS project also aims to achieve:

> *The definition of a viable Roadmap of an evolution path for today's network models, protocols (e.f. IPv6) and paradigms, as guided by the GANA reference model.*

To accomplish this matter six main objectives were identified that would be used as guidance for the evolution of the project. The marked objectives of the problem were [11]:

- *Objective 1*: Specification of some of the Autonomic Behaviours to be implemented in different networking environments, such as self-adaptive routing in the core network.
- *Objective 2*: Examination and identification of those existing characteristics related to the IPv6 protocols that can be exploitable to be used in the development of the Autonomic Behaviours.

- *Objective 3*: Investigation and creation of the IPv6 protocol extensions that are necessary for implementing the different Autonomic Behaviours defined.
- *Objective 4*: Investigation and creation of the network components, algorithms and paradigms necessary for implementing the different Autonomic Behaviours defined.
- *Objective 5*: Selection of those Autonomic Behaviours among the defined ones to be implemented and demonstrated in a testbed scenario.
- *Objective 6*: Industrialization and Standardization of the Autonomic Behaviour Specifications (ABs) and the protocol extensions with the help of the standardization bodies.

Within the objectives defined by EFIPSANS, this thesis work is scoped into the "Objective 4". We have contributed to the investigation and creation of techniques to be applied during the design period of the GANA elements and the Autonomic Behaviours in the autonomic networks. We offer an extended discussion about the contributions in Chapters 3 and 4.

2.2 GANA Reference Model

It has been previously mentioned that prior to the EFIPSANS project there was a lack of a generic reference model for the development of autonomic networks. This was one of the main reasons for launching the EFIPSANS project, and it became one of the main purposes of it. EFIPSANS has been developing a Generic Autonomic Network Architecture (GANA) to be used as a holistic Reference Model for Autonomic Network Engineering, in other words, for Self-Management within the Node and Network Architectures [7]. Although the EFIPSANS project is based on an evolvable approach of the existing infrastructure, the aim has been that GANA would be a valid reference model for both revolutionary and evolutionary approaches. GANA would allow the standardization of Autonomic Behaviour Specifications (ABs), which are the functions achieving the self-managed characteristics.

The GANA architecture is the framework for developing Autonomic Behaviours, thus its characteristics have been used along the process of this thesis.

2.2.1 GANA Concepts

In the following paragraphs we will introduce a number of key terms important for discussing the concepts of GANA, such as the mechanisms allowing the development of the Autonomic Behaviours or the different architectural elements [7].

Decision-Making Element / Decision Element

The Decision-Making Element (DME), also referred to as Decision Element (DE), is the element in the GANA architecture that assumes the role of the autonomic manager in certain behaviour considered as autonomic [7]. These elements trigger the needed changes in the Managed Entities driving to process the required behaviour. Basically, the DE is responsible for making the changes in a subordinate Managed Entity (ME), based on an internal control-loop that creates networking decisions. The changes produced by the decisions have the final purpose of driving the network environments towards certain networking behaviours.

Managed Entity

The Managed Entity (ME) is a resource controlled by the DE that can directly affect certain characteristics of the network. The MEs in the network are those elements that will change the parameters/characteristics according to the Autonomic Behaviour rules in order to achieve the required automated task. In GANA the concept of "entity" refers to both physical and abstract characteristics of the networks, trying to avoid the established concepts of a physical device as the unique element to manage [14].

Control Loops

A Control Loop in GANA is an internal functional loop of the DE aiming to properly control the ME and its characteristics, making the autonomic tasks conform to the policy established for achieving the required network behaviour. The control loop in a DE is fed with (feedback) information from different dis-

tributed information suppliers of the network, such as ME or network probes. Depending on different policies or goals established by the administrator of the network the DE analyses the information and makes a networking decision. This decision will be passed down to the managed entities as a command for executing a change in the behaviour. Once the change is done on the ME, the DE continues to be fed with new information representing the current condition of the network, starting the control loop again. In Figure 2.1 the generic model of a control loop defined by GANA is shown [14].

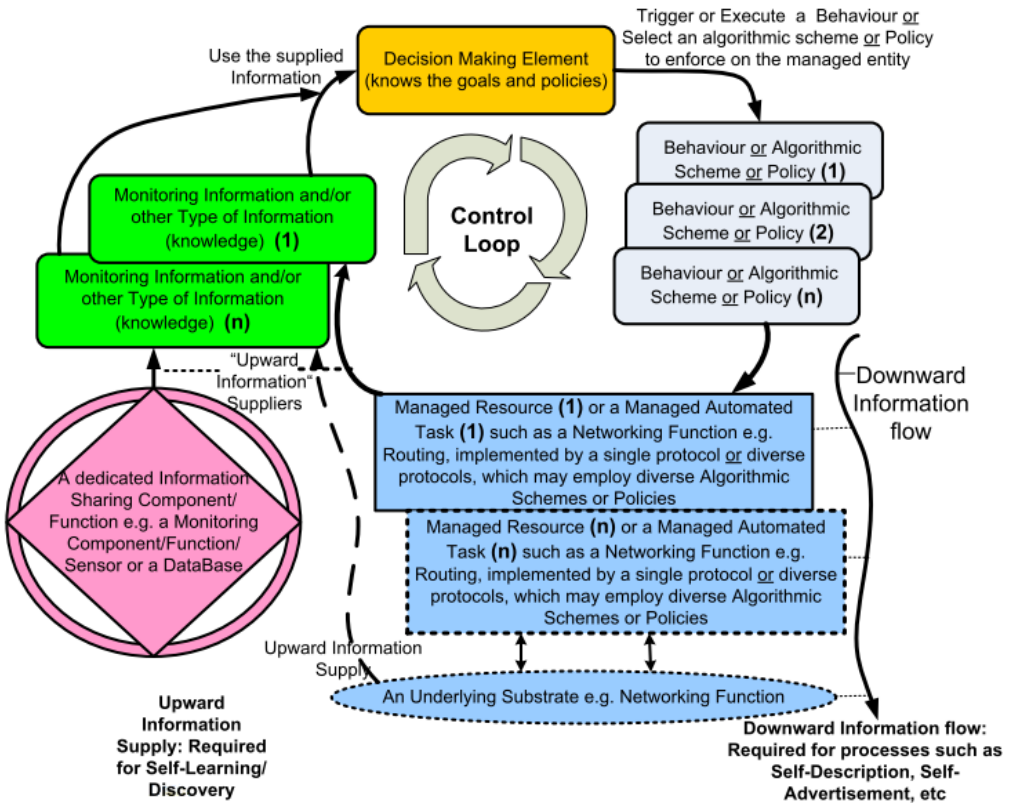

Figure 2.1: Generic Model of Abstract Autonomic Networked System [11]

Autonomic Behaviour

An Autonomic Behaviour (AB) is the behaviour that rules certain elements or entities in a network, whose final purpose is to define some autonomicity characteristic; in other words, that the desired self-managed behaviour is automatically reached thanks to the communication between elements and without the

necessity of human intervention. Examples of ABs are self-description, self-advertisement, self-healing, self-configuration, etc [14].

The concept of AB involves all those sub-behaviours sharing the common goal introduced into the network, and which are running or being executed at the MEs. Besides, in an AB the way the DEs have been fed with information is also specified, the type of this information and which were the mechanisms that caused the final decision to be triggered into the MEs by the DEs. Furthermore, policies or profiles introduced into the network by the administrator also form part of the AB concept. These policies/profiles will rule the global behaviour required by the autonomic network. Further description and information about Autonomic Behaviours is covered in Section 2.3

2.2.2 GANA Principles

Several ideas from different projects studying autonomic networks, such as 4D [8] and CONMan [9], were brought into the GANA architecture, in particular the characteristics of the functional planes. However, the GANA architecture goes one step beyond this as it considers the necessary generic principles for creating an evolvable autonomic network. This is achieved by defining the different network elements involved in the autonomic tasks, the hierarchical functional abstractions of the network, the self-manageability aspects, the needed interfaces allowing new components to be integrated into the architecture, and the relationship between network elements and the rest of the network architecture [13].

One of the main benefits of the GANA architecture is that it allows the creation of Autonomic Behaviour Specifications, which can be standardized in the standardization bodies. The main reason for this characteristic is a principle followed during the project [14], which states: " *Clearly separate specification issues for autonomic behaviours (that includes the structural and behavioural aspects of the associated Decision-Elements) from their implementation issues"*. This conceptual principle has helped to clearly address several of the problems that were identified as potential issues to come up during the designing progress of GANA. These identified issues are described in [14] and are:

- *Complexity*: the four hierarchical levels of abstraction allow addressing the complexity of the architecture by dividing the network functionalities.
- *Conflict-free and stable decisions*: the GANA architecture must ensure that the decisions made by the DE are stable avoiding any kind of decision confliction.
- *In-network management*: the self-manageable tasks should be designed in a way that the constrains and the boundaries for the in-network management are established, avoiding the external intervention in the management tasks.
- *Interfaces for Network Governance*: in the GANA architecture it is also defined the way operators or network administrators will be able to set the final objectives that will govern the autonomic network behaviours. This is done by operator policies or defined profiles.

The defined hierarchy at different levels of abstraction for the control loops is a key principle. This control hierarchy provides the architecture with a coordinated access-control to the MEs, allowing the synchronization of the communication between the involved DEs or other entities. This coordination will drive the changes in a way that no conflicts appear between different Autonomic Behaviours at the different levels of abstraction.

This principle has been studied and imported from the control theory field. In this field, the hierarchy intends to decouple the control systems working at different operating levels and managing independent information, to finally impose the control tasks at different timescales [14].

2.2.3 GANA Architecture

In this section the elements of the GANA architecture that are relevant for the study of this work are discussed. These elements will be later used during the progress of the thesis.

Hierarchical Control-Loops

In GANA four levels of abstraction were identified in order to clearly separate the structural and behavioural aspects of an Autonomic Behaviour. These levels represent the four abstract levels in which a network is conceptually divided. These four levels build the GANA Hierarchical Control Loop (HCLs) framework, and are defined as:

- *Level-1 - Protocol-level*: This is the lowest level and represents the network protocols and their parameters. At this level normally lie the lowest MEs controlled by upper DEs. However, in most of the cases a protocol may include any kind of control-loop in its core logic, allowing to modify its own characteristics. In this cases, the protocol level abstraction is considered as a **Protocol-Level-DE**. Otherwise, any manageable characteristic of a networking protocol, such as routing tables, are considered as **Protocol-Level-MEs**.

- *Level-2 - Abstracted network functions-level*: This level lies above the protocol-level and represents an abstracted network function, such as forwarding, routing, QoS management or mobility management.

  The **Function-Level-DE** is the DE that manages a collection of protocols belonging to certain Function Block. A Function Block represents an abstracted network functionality as previously mentioned. On this level DEs are able to change the characteristics of all those protocols involved in the abstract functionality, driving the behaviour of them to the unified required behaviour.

- *Level-3 - Node/device's overall functionality and behaviour level*: This layer represents the entire functionality of a physical node or system in the network.

  The **Node-Main-DE** is the DE managing the collection of all the Function Blocks, as described in the previous paragraph, ruling the different networking behaviours of a device.

- *Level-4 - Network's overall functionality and behaviour level*: This is the highest level control-loop. This layer represents the behaviour of the whole network, in other words, all the devices that make up the autonomic network.

Several **Network-Level-DEs** govern the whole network behaviour from the centralized point of view. These kinds of DEs know the main policies and goals established by the network administrator, and which decisions will drive the whole network characteristics and behaviours. These DEs are separated by conceptual characteristics of the network, such as Routing Management (Net-Level-RM-DE) or Quality of Service (QoS) management (Net-Level-QoS-DE).

It is important to note that the concept of ME not only involves those protocol-level entities receiving a command from a Function-Level-DE, but also involves those DEs running at a lower hierarchical level of abstraction that are managed by any upper DE. For example, the Node-Main-DE is the ME of all the Network-Level-DEs [14].

Each of the introduced control-loops has its equivalent conceptual representation from the generic one represented in Figure 2.1.

As it is indicated in the principles of GANA, the hierarchical control loops offer a great flexibility for controlling different levels of abstraction independently, something that leads addressing the design complexity of the Autonomic Behaviours in an easier way.

Communication between Control Loops

In the GANA architecture different kinds of relationships have been defined to be established between the entities. Depending on the relationship of a DE with other DEs, the role of that DE is then determined. This role will describe the characteristics of the DE in the hierarchical structure, and the type of communication that is allowed with other entities. The defined relationship types in GANA are [14]:

- *Hierarchical Relationship*: this is the normal hierarchical relationship between two components in adjacent hierarchical levels. One DE has a hierarchical relationship with its lower level DE/ME and with its immediate upper level DE.

- *Peer Relationship*: this is the relationship between DEs for exchanging information about the DE's configuration or for requesting some services between DEs.
- *Sibling Relationship*: this is the relationship between two DEs of the same level that allows creating peering relationships with the managed entities owned by another DE.

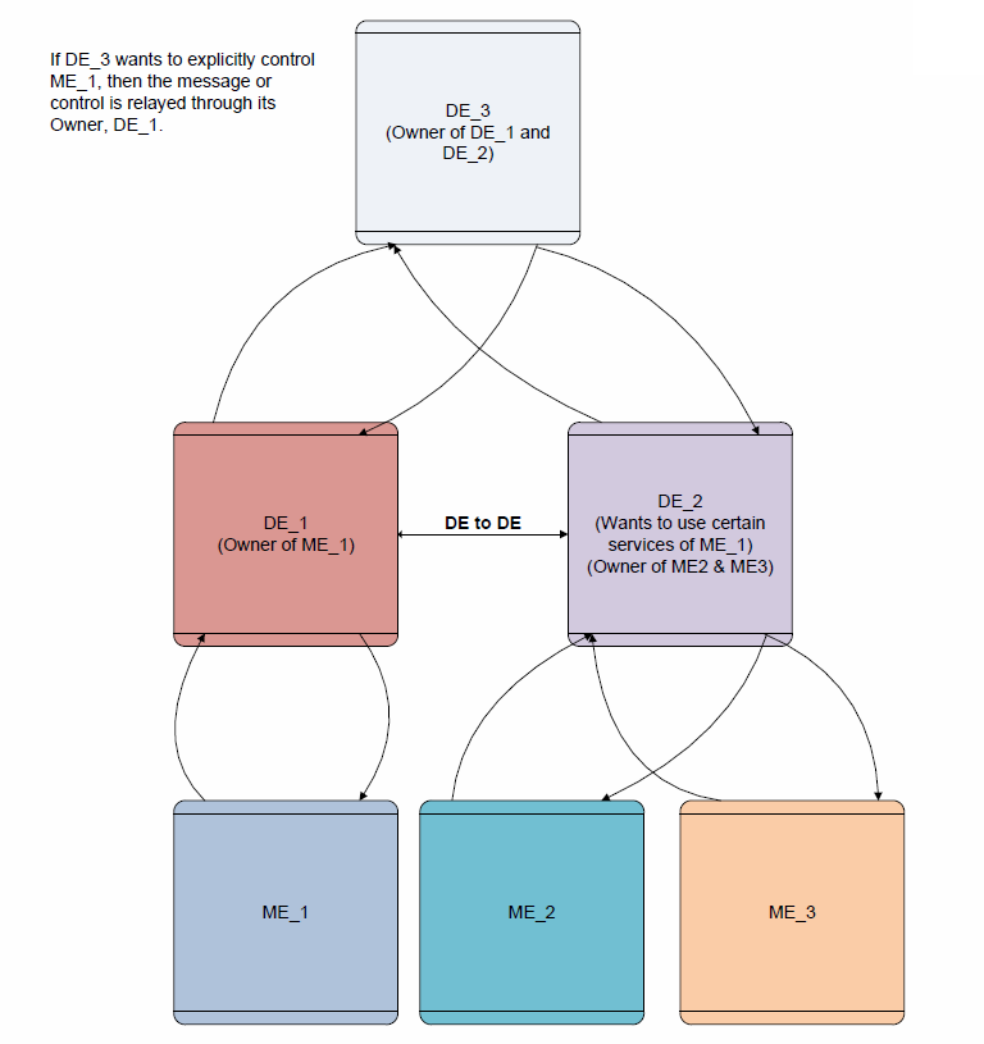

Figure 2.2: Different relationships between DEs and MEs

GANA has a notion of "ownership". In the GANA architecture is stated that just one ME can be managed by one DE. In case one DE wants to start any

process in a ME owned by other DE, it must first establish a sibling/peering relationship between both DEs [15]. This is shown in Figure 2.2.

Functional Planes

The functional planes of today's networks are divided into different categories, such as the control plane or knowledge plane. Some other autonomic projects, such as 4D [8] and CONman [9], concluded that the current division was confusing and not useful for the design of the future internet. Both projects shared the ideas that today's functional planes could be compressed into four functional planes: Decision plane, Dissemination plane, Discovery Plane and Data plane.

During the creation of the GANA reference model, these ideas were considered and it was concluded that they fulfil the majority of the necessities observed in this matter. Thus, the GANA reference model, in order to benefit both discussed approaches for designing the future internet (evolution and revolution), decided to adopt these ideas [8] with some changes. The functional planes in the GANA architecture are as follows [14]:

Decision Plane In this plane all the decisions that would drive or change the behaviours of the elements in the network are included. The events creating the different decisions happen in real-time manner, thus evaluating the network's traffic, topology changes, etc.

Obviously, the elements of this functional plane are the ones creating and triggering the decisions: the Decision Elements (DE).

Dissemination Plane The Dissemination Plane is formed by the collection of mechanisms and protocols that allows the exchange of non-user traffic information between the elements of the architecture, both inside a node or between nodes. This kind of information covers control information, signalling information, monitoring data, alarms, events, etc.

The DEs make use of the Dissemination Plane to exchange information between them. Some of the protocols considered of this plane are: ICMPv6, DHCPv6 or SNMP.

Discovery Plane The Discovery Plane is formed by the collection of mechanisms and protocols responsible for the discovery of the entities integrated and associated to the network. In this plane the management implied in the discovery procedure is done automatically. In addition, during this automatic task the type of relationships between elements or the assigned capabilities are defined.

Each DE has an associated Discovery Plane that will inform the others and the network of its presence and own characteristics. Examples of procedures in this plane are: IPv6 Neighbour Discovery, Topology-Discovery Protocols, etc.

Data Plane The Data Plane is formed by the collection of mechanisms and protocols that manage the user traffic. In addition, this plane defines how the entities should process the user traffic information according to the decisions made by the DEs. Examples of this plane are the TCP and UDP protocols, and IP forwarding, forwarding tables, packet filters, etc.

Elements of a Control Loop

There are three main groups of elements that form any Control Loop: the DE, the ME and the monitoring and information gathering devices, all of them having different interfaces for different purposes.

Figure 2.3 shows the graphical representation of a DE and its interfaces. This general representation of a DE represents also intermediate MEs, which are at the same time DEs of underlying MEs. The case when the ME is the lowest level ME (the one that does not have an underlying MEs, i.e. the protocol level ME) is presented in Figure 2.4. In all these entities there are three interfaces forming the management interface: the sensory interface, the effectors interface and the general non-sensory for information retrieval interface. These interfaces are used to answer the corresponding sensory, effectors or general non-sensory calls from the upper DEs [7].

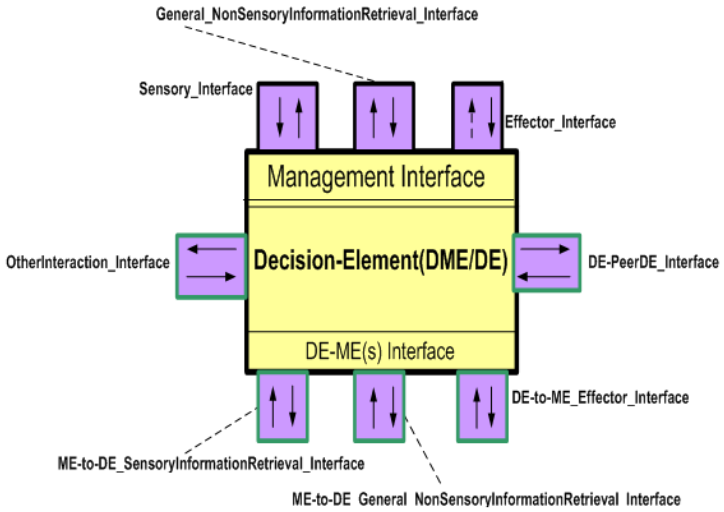

Figure 2.3: General representation of the interfaces of a DE [7]

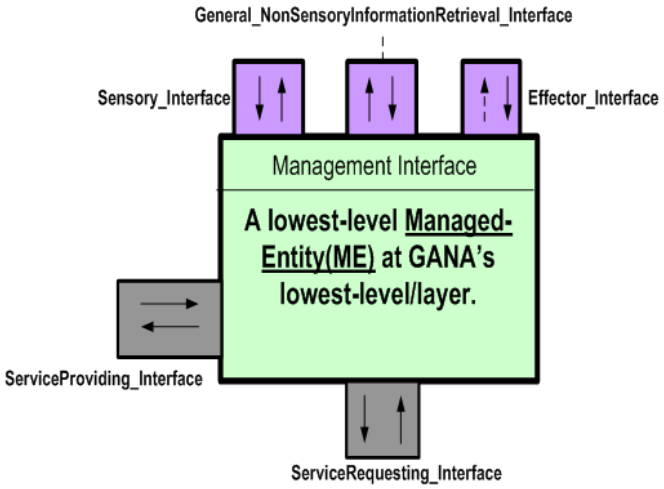

Figure 2.4: General representation of the lowest level ME [7]

In the DE three other interfaces form the DE-to-ME interface: the sensory, effectors and general non-sensory interfaces. The DE will use the sensory interface to initiate the retrieval of information related to a specific Autonomic Behaviour with the ME, which will respond via this interface with an answer to the request. The effectors interface is used by the DE to trigger an execution in the managed ME, such as starting a timer or enforcing a policy [7]. The gen-

eral non-sensory information retrieval interface is used to get information from the ME that is not related to the specific Autonomic Behaviour.

In addition, the DEs have two other interfaces: the DE-to-DE interface for establishing the sibling and peering communications, and the other-interaction interface, whose purpose is to communicate with any other entity that is not an owned ME or a peer/sibling DE, such as information suppliers or service providers. The MEs have also two additional interfaces: the service providing interface, where an external entity can request a service from; and the service requesting interface for requesting services to other MEs.

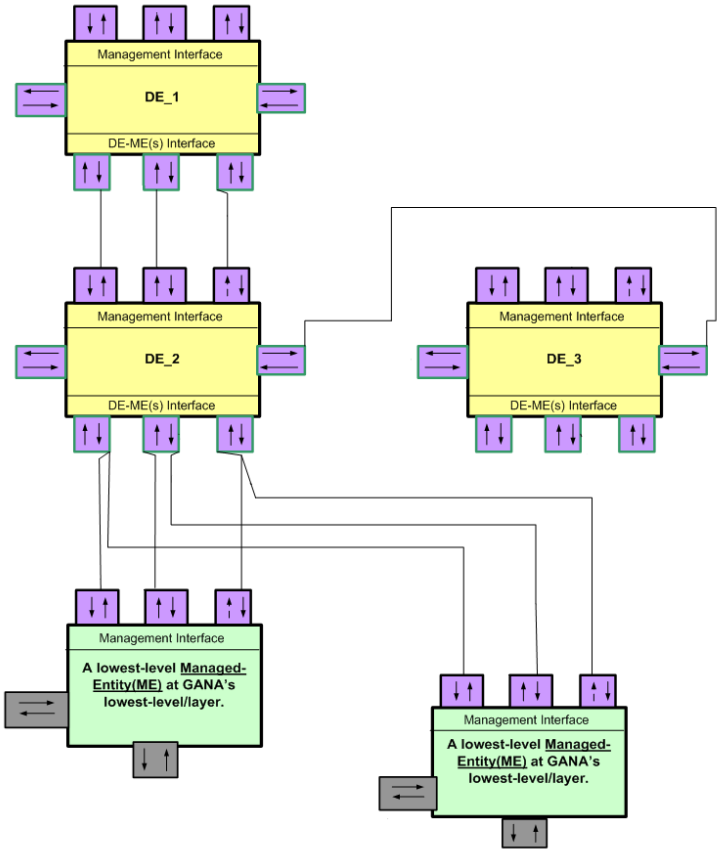

Figure 2.5: Interconnections between DEs and MEs [7]

Overlay Network for Information eXchange

One additional element (or group of elements) needs to be included into the GANA architecture. This element is known as the Overlay Network for Information eXchange (ONIX) and is responsible for interacting between the network administrators and the rest of the entities [16].

ONIX is a distributed collection of servers or systems whose purpose is to have an overall overview of the network elements capabilities and services offered by them. It will be the system responsible for pushing or publishing this information into other elements requesting them. In addition, it is the system where the policies, goals and profiles defined by the administrator are firstly stored. As a consequence, one of its main roles is to distribute this information to the rest of the elements (such as DEs) making them aware of the administrator's network behaviour wishes. In this thesis the profile or policy definition techniques are not covered.

2.3 Autonomic Behaviours

As discussed previously, an Autonomic Behaviour (AB) is the collection of behaviours or sub-behaviours that will drive the entities of a network to reach a final desired goal. These behaviours are managed by the network's own entities, and thereby creating a self-manageable network [14].

In this section we provide a brief introduction to the different Autonomic Behaviours scenarios studied within the EFIPSANS project. We intend to show several existing possibilities for designing and developing Autonomic Behaviour by using the concepts explained up to this point. The Autonomic Behaviours scenarios identified and developed during the EFIPSANS project are [17]:

- *Self-Configuration of Routers using Routing Profiles in Fixed Network Environment*: This scenario demonstrates how a device initially connected to the network can be auto-configured thanks to the auto-discovery and auto-configuration processes. The devices will obtain the specific role to be played in the network.

- *Auto-Collaboration for Optimal Network Resource Utilization in a Fixed Networks*: In this scenario is demonstrated how self-adaptation and self-optimization of resources can be achieved through other self-* functionalities, such as self-description.

- *Auto-Configuration / Self-Configuration of Addresses in a SOHO Network*: This scenario shows how IPv6 addresses in a Small Office/Home Office (SOHO) network can be dynamically configured using the auto-configuration capability.

- *Auto-Configuration / Self-Configuration of Addresses in MARSIAN Platform*: This scenario has demonstrated how some self-* functionalities can be used in Mobile Ad hoc NETworks (MANET) to create enhanced networks known as MARSIAN (Management, Auto-configuration, Resilience and Survivability In Mobile Ad hoc Networks).

- *Auto-Configuration of Radio Channels in 802.11 networks*: According to this scenario auto-configuration can increase the performance and reliability of wireless networks by dynamically selecting the best radio channels and performing proper channel reallocations.

- *Autonomic multipath routing in 802.11 Mesh Networks*: GANA auto-configuration characteristics can be used to establish multi-path-routing capable Ad hoc networks.

- *Autonomic Routing and Self-Adaptation driven by Risk-Level Assessment in Fixed Network Environments*: This scenario demonstrates how Autonomic Routing and Self-Adaptation can be used to address potential failures on the network.

- *Autonomic Mobility and QoS Management over an Integrated Heterogeneous Wireless Environment*: Functionalities of autonomic mobility and QoS management can be used to allow heterogeneous environments working together in a collaborative manner, as demonstrated in this scenario.

- *Autonomic QoS Management in Wired Network*: Network performances can also be enhanced in a fixed network by controlling the QoS behaviour with self-adaptation and self-configuration capabilities.

- *Autonomic Peer-To-Peer (p2p) Network Monitoring Scenario*: In this scenario is demonstrated how some Self-configuration, Self-organization, Self-

healing and Self-optimisation capabilities can be introduced into MANETs through the exploitation of P2P techniques.

- *Network Monitoring and QoS Management Scenario*: Collaborative monitoring procedures can enhance the QoS in a network. This monitoring system would collect information from different protocols and represent the results against certain shared behaviours.
- *Autonomic Fault-Management for selected types of Black Holes in a Fixed Network*: This scenario demonstrates how "silent" networking faults know as "Black Holes" can be solved by the Reactive Resilience and Autonomic Fault-Management self-* functionalities.

These twelve scenarios briefly introduced have been demonstrated within the EFIPSANS project. These scenarios were integrated and run in several testbeds, this provided a huge insight to the project's outcome.

2.4 Summary

In this chapter we have introduced the EFIPSANS project and the autonomic networks concept. The autonomicity in the networks defined by EFIPSANS is reached by self-manageable functionalities known as Autonomic Behaviours. These Autonomic Behaviours are based on the GANA architecture, a holistic reference model for the development of autonomic network.

Decision Elements (DE), Management Elements (ME), ONIX and hierarchical control loops are the introduced concepts which make up the GANA architecture. These concepts are involved in the design of Autonomic Behaviours, thus it is important to remember their role for achieving autonomic functionalities. The GANA architecture is divided in four levels of abstraction (Protocol, Function, Node and Network), where different control loops running at the DEs will trigger some changes into subordinated MEs for achieving a common goal.

In Chapter 5 these concepts are used for designing the structure of an Autonomic Behaviour, which will be used for studying the applicability of a Model-Driven Methodology for the design of GANA Autonomic Behaviours.

Chapter 3

Modelling the GANA Architecture

In this Chapter the concept behind the Model-Driven Methodology to be applied in the development of GANA elements is studied. In addition, we discuss the necessary modelling requirements and the GANA Meta-Model to be used in the modelling of the Autonomic Behaviours. One of the main parts of our contribution to the EFIPSANS project is the development of software applications managing GANA models as data structures. For such a purpose, we have studied the Eclipse Modelling Framework (EMF), and thus, a short introduction of this technology is included in the Chapter.

By applying a model-driven methodology we aim to address some problems that may appear during the design period of control loops. One of the most relevant issues that is intended to be addressed by using the methodology is the *stability* of the control loops and the behaviours applied by the DEs. Addressing stability aspects of such complex networks can be a difficult task, however, some of the stability aspects can be investigated during the design period by evaluating and verifying the capabilities of the GANA systems. This investigation can be carried out using the services offered by the model-driven methodology proposed, which must include modelling properties of both structural and behavioural aspects.

3.1 Introduction to Model-Driven Methodologies

Creating software applications for complex systems has been a difficult task for many years. This complexity associated with the design and development of software to be run on different and distributed systems has called for some methods to be used by engineers that would ease the production of these systems. Within the Software Engineering concept many different solutions have been proposed to address the complexity. As it is defined by Roger S. Pressman in his book "*Software Engineering: A Practitioner's approach*" [18] Software Engineering is:

> *Software Engineering practice is a broad array of principles, concepts, methods, and tools that you must consider as software is planned and developed. Principles that guide practice establish a foundation form which software engineering is conducted.*

One of these Software Engineering methods is called Model-Driven Engineering (MDE). Model-Driven Engineering is the practice aiming to design models of the software, which resemble its characteristics and architecture as a whole. Even though the "software", as such, is an abstract material, similar techniques already used in the development of physical entities should be applied to its own development process. If we consider the development of a car, it is known that different models representing the car will be created, such as the scaled replica, the blueprints of the pieces, the electric architecture, etc. Similar approaches are needed for the design of software systems, which models would guide and help engineers to properly develop and code the entities to be created.

Those engineers, within the Software Engineering process, creating and using different models for the development of software systems are known to follow a Model-Driven Methodology. In other words, Model-Driven Methodologies are based on the usage of models for designing, simulating, verifying, validating and testing some of the software characteristics during the development of a product [19].

As previously discussed in Section 2.1.2, one of the main purposes of the EFIP-SANS project is to develop a reference model to be used for the development

of autonomic networks, in other words, a model describing the whole GANA architecture. Furthermore, within the project we have aimed to investigate a Model-Driven Methodology to be applied in the development of GANA Autonomic Behaviours by defining the GANA Meta-Model. Part of our contribution refers to the GANA Meta-Model characteristics and its usage.

3.1.1 Model Levels

There are three levels of models involved in the modelling process of a Model-Driven Methodology: meta-meta-models, meta-models and models. These three concepts depend of each other for creating a three-level stack, where models are the bottom level and meta-meta-models the top level. Even thought they are named differently, it should be noted that the three are just models. These concepts are further described in the following paragraphs from a top-down approach. After this, a subsection describes an example for clarification purposes. Finally, we explain in the last part of this section some benefits that this modelling approach offers. In the next section, the Eclipse Modelling Framework (EMF) and its meta-meta-model (Ecore) are further discussed with a software example.

Meta-Meta-Models

Meta-meta-models are the collection of elements and characteristics used for the creation of the meta-models that will describe a system. These meta-meta-models have a certain level of standardization as they are provided by Modelling Frameworks of the market or by standardization bodies, such as the Meta-Object Facility (MOF) standard defined by Object Management Group (OMG) [20]. Two different meta-meta-models from two Modelling Frameworks need to be mentioned as they are used in the context of this work: Ecore from EMF and MetaGME from Generic Modelling Environment (GME) [23]. We provide a brief introduction to Ecore from EMF in Section 3.2.1; however, MetaGME is not further discussed as the usage of the GME application is based on the creation of models by using the already created GANA Meta-Model.

Meta-Models

Meta-models are the models that abstractly represent some characteristic of the system to be developed, such as the structure or the behaviour. Meta-models can be considered as the blueprints of the system and they are created with the elements and tools provided by the Modelling Frameworks, such as EMF or GME. Meta-models conform to their meta-meta-models. This means that a meta-model can be described using the tools defined in the meta-meta-model. Basically we can surmise that if a meta-model has been designed with the elements provided by the Modelling Framework (meta-meta-model), then the meta-model conforms to the meta-meta-model of that particular Modelling Framework.

Models

The lowest-level in the model-stack is defined as "models". A model can be any artefact representing a conceptual part of a real system or the whole real system itself. So basically, this level of models includes, for example, behavioural representations with Finite State Machines (FSM) or Message Sequence Charts (MSC); or structural characteristics written in Java code.

Systems

The three modelling levels are used for designing our systems. The "models" created will represent the system in different ways, and they can be used, for example, for testing purposes. A final developed product is considered a "system", as it is not a representation any more. In other words, a "model" representing a concrete data structure during the design period becomes a "system" at run-time in an application.

Example of the three models

The following example aims to clarify some aspects of the modelling levels by using the car case previously mentioned. We mentioned that when designing a car three different models could be used: the scaled replica, the blueprints or

the electrical architecture. These three models represent the car in an abstract way, so their definitions are considered the parts of the car's meta-model. We would have then three different parts of the general meta-model for creating models representing our system.

We can consider the scaled replica as the meta-model part representing the outside structure of our car. Thus, if we want to build the outside structure (model) of the real car (system) we would use the designed scaled replica (meta-model) as the reference model during the development process.

The blueprints are the meta-model part describing the structure of all the pieces (models) of our car. Some of the structures could have variable or fixed parameters, which will be personalized for each model created. For example, within these blueprints we could specify a fixed size of the wheels while specifying a variable texture/colour of the seats. These blueprints (meta-models) will be used as the reference model when creating the parts (models) of a car; therefore, we could create a car with fixed wheels size but with variable texture/colour of seats attending the buyer's demands.

In addition, the electrical architecture design would be the meta-model part describing the behaviour of the electrical construction. In this meta-model we could specify the obligatory electrical parts and several variable/interchangeable parts, such as selecting between normal lights or neon lights. When creating the electrical architecture of a car we will follow the rules defined in the meta-model, so for example, adding the obligatory parts and neon lights as a variable part. The resulting electrical architecture (model) will be conforming to the rules of the meta-model.

The whole car meta-model will gather at least these three meta-model parts; therefore, we have three different ways of creating models representing our car. If we want to develop the whole car (system), which is formed from different models, we have to follow all the rules within each part of the meta-model.

Even though three very different parts have built the car's meta-model, a unique meta-meta-model is used for defining all those parts. This implies that the meta-meta-model used for creating the car's meta-model is formed by the collection of parameters and tools used for creating the different parts of the meta-model. For example, the meta-meta-model could group the elements cork and knife in the scale-replica folder, the condenser and coil representations in the

electric folder, and the size or colour parameters in the blueprint folder. Thus, we would use these tools of the different folders to create the different parts of our meta-model. In addition, the meta-meta-model would define how elements in a folder interact with each other, for example, in the scale-replica folder is defined that the knife can cut the cork; and how elements of different folders are related to each other, such as negating the interaction of elements from different folders.

Finally, based on our meta-model and its parts we could create for testing purposes parts of the car (models), such as a door or the electric framework; or even the whole car (system). Each of these created parts, which are based on the defined meta-model, are the actual models. This entails that the resulting models conform to the meta-model, as they have been created based uniquely on the defined meta-model. In other words, we would not be able to create a car with four doors if our designed meta-model had just two doors. This would mean that our models would no longer conform to the designed meta-model.

Let's consider now a simple GANA architecture example. Using one meta-meta-model offered by any of the mentioned Modelling Frameworks (Ecore by EMF or MetaGME by GME) the GANA Meta-Model is defined. We can create a model representing the software of a router by following the GANA Meta-Model guidelines. This model would have, for example, certain interface names as defined in the meta-model. It would be an incorrect practice if we introduce a new interface name that was not defined in the GANA Meta-Model, as the created model would not conform to the GANA Meta-Model anymore. The software created and running at the router would be considered the system.

Benefits of the three-level modelling approach

One benefit that this three-level modelling approach offers is the design of a system in a structured way [21]. First, the meta-model is created according to some system requirements. In this step, architectural experts can introduce, for example, necessary characteristics for overcoming stability issues. Later, testing engineers can create models representing their system without having to care about architectural matters.

There are further benefits in the case of using a well known modelling framework and its meta-meta-models for the modelling of the systems. For exam-

ple, the models can be exchanged between different applications supporting the shared language. In addition, modelling frameworks, such as EMF, offer partial code-generation from the structural definition of the models [21].

3.2 Eclipse Modelling Framework

There is a known polarity division between software engineers developing applications: requirements engineering and modelling engineering [18]. On one side, engineers following the requirements engineering approach develop software applications basing the development process on the requirements established. These engineers may use some models along the development process; however, it could be that they do no use models at all. On the other side, engineers following the modelling engineering approach tend to use models representing their systems along the development process of the applications. The Eclipse foundation has created a modelling framework for developing software applications based on the usage of explicit modelling languages. This framework, know as Eclipse Modelling Framework (EMF), provides the engineers with tools for the creation of meta-models and models representing the software systems [21].

As previously mentioned, an important part of this work is based on the creation of software applications to be used in the tool-chain described in Chapter 4. Due to the nature of the requirements, the applications were developed based on the modelling approach. This means using models as data structures to be modified or processed within the applications.

The meta-meta-model that EMF provides is called *Ecore*. By understanding the EMF architecture and the Ecore meta-meta-model we can create, modify and process models representing our systems. In order to properly understand Ecore, a certain knowledge of UML diagrams is needed.

3.2.1 Ecore Meta-Meta-Model

Ecore is the meta-meta-model used in the Eclipse Modelling Framework. It is a Java library providing the needed tools for creating the meta-models which describe the systems. The tools that Ecore includes are different Java classes

and the relationship that these classes have between each other. By using different classes of the Ecore library we can design the meta-models representing our systems. In Figure 3.1 the different classes that Ecore offers are shown.

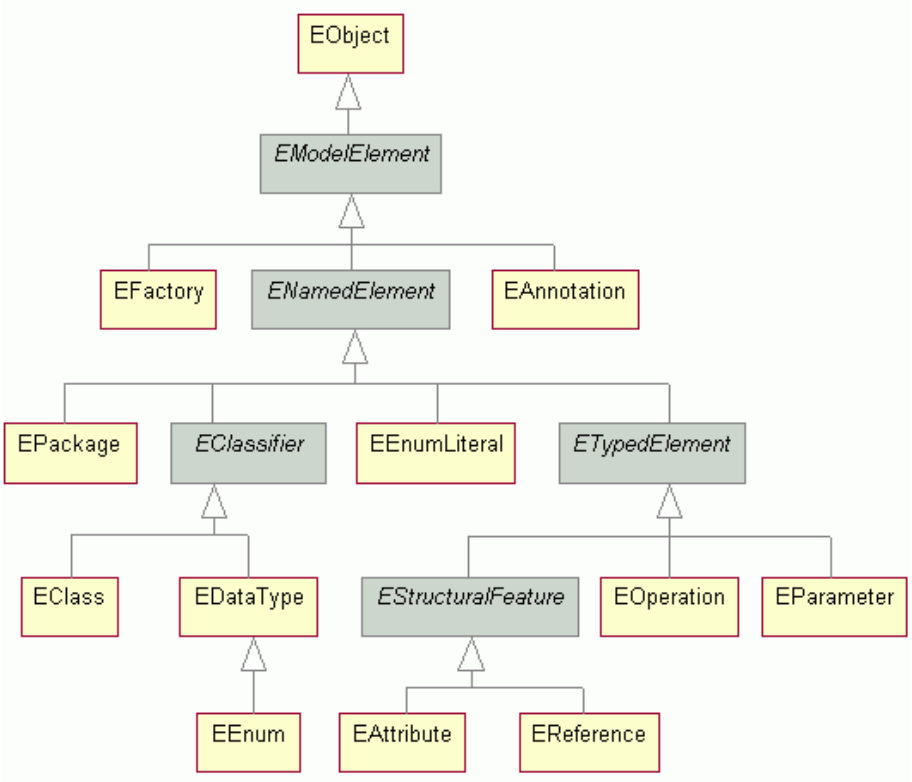

Figure 3.1: Complete class hierarchy of the Ecore model [22]

A simplified example of four of these classes and the relationship between them is shown in Figure 3.2. These four classes can be considered the most relevant classes of the Ecore library, as many meta-models can be created based on them. The meanings of them are [21]:

- *EClass*: it represents a modelled class. As it is shown in Figure 3.2 it has a name, and can have zero or more attributes and zero or more references.
- *EAttribute*: it represents a modelled attribute. As seen in the figure, attributes have a name and a data type.
- *EReference*: it represents a modelled association to other class. It has a name and a boolean attributes indicating containment of the class.

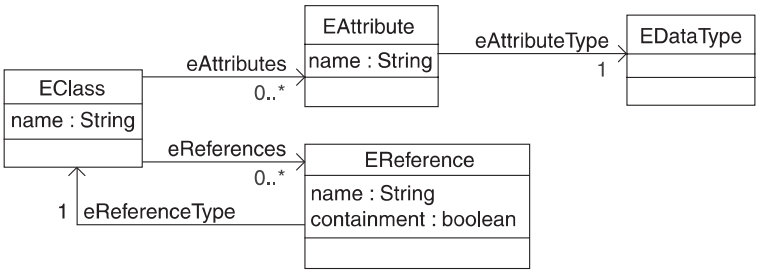

Figure 3.2: Part of the Ecore model [21]

- *EDataType*: it represents the data type of the attributes. The range of the data types is defined by the Java primitive data types, in other words, *int*, *String*, *float*, *Date*, etc

These four classes of the meta-meta-model can be used for creating complex meta-models. From the software structure point of view, these classes can be used for modelling the classes of the system to be developed.

In order to create an example, let's imagine that we want to design an application handling CDs of mixed music, where different artists and songs are included. Each artist can have many songs, and each song can be sung by one or more artists. This very simple example is represented with UML language in Figure 3.3.

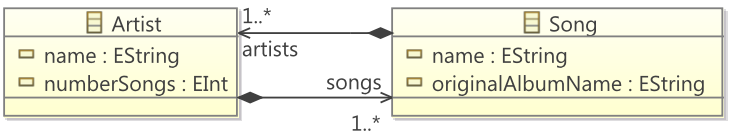

Figure 3.3: Meta-Model of artist-song model

In the UML diagram we can observe that there are two classes: the songs and the artists. To conform to the Ecore model (meta-meta-model) the classes are from the type *EClass* and the names associated to them are *Artist* and *Song*. The EClass *Artist* have two EAttributes: the *name* of the artist, which EDataType is *String* and *numberSongs*, which EDataType is *int*. On the other

hand, the EClass *Song* has two EAttributes: the *name* and the *originalAlbumName*, which both have EDataType as *String*. In addition, EClass *Artist* has an EReference *songs* to the EClass *Song* that refers to all the songs from that artist, which can be 1 or more; and the EClass *Song* has an EReference *artists* that refers to the artists forming part of a song, which can be one or more.

The UML diagram in Figure 3.3 is considered the abstract meta-model representation of a CD. From this meta-model we could create models representing CDs, e.g. an entry in a database representing a CD with added track names and artist names. In case one of these CDs is physically created we could consider it as a "system". Let's imagine a software application handling CDs of music for seeing this example from a software perspective. This application could contain the meta-model of the CD to know how they are structured. In addition, an internal database would contain several models representing the CDs with tracks names and artist names. The actual tracks stored in other database would be considered as the "system", as they are not a representation of the information anymore.

Serialization of the model

One of the biggest advantages offered by EMF is the serialization of the models based on Ecore. This serialization will create XML (Extensible Markup Language) files containing the information of the models, which can be exported to any other application understanding the Ecore meta-meta-model. So obviously, in order to understand the models by an internal or external application, both the meta-model created together with the system model should be exchanged.

The Ecore based models are serialized in XML based files known as XML Metadata Interchange (XMI). These files can be used, for example, to import the structural model in a code generation program that would create the skeleton of our system, i.e. the Java classes, the attributes and some methods for accessing them. This service is offered by the EMF framework [21]. In the following lines the serialization of the meta-model created is shown.

```
<?xml version="1.0" encoding="UTF-8"?>
<ecore:EPackage xmi:version="2.0"
   xmlns:xmi="http://www.omg.org/XMI" xmlns:xsi="http://www.w3.org/2001/
              XMLSchema-instance"
```

```
   xmlns:ecore="http://www.eclipse.org/emf/2002/Ecore" name="cd"
   nsURI="http:///com/thesis/cd.ecore" nsPrefix="com.thesis.cd">
 <eClassifiers xsi:type="ecore:EClass" name="Artist">
   <eStructuralFeatures xsi:type="ecore:EAttribute" name="name" eType=
     "ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
   <eStructuralFeatures xsi:type="ecore:EAttribute" name="numberSongs"
     eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EInt"/>
   <eStructuralFeatures xsi:type="ecore:EReference" name="songs"
     lowerBound="1" upperBound="-1"
       eType="#//Song" containment="true" resolveProxies="false"/>
 </eClassifiers>
 <eClassifiers xsi:type="ecore:EClass" name="Song">
   <eOperations name="originalAlbumName" eType="ecore:EDataType
      http://www.eclipse.org/emf/2002/Ecore#//EString"/>
   <eStructuralFeatures xsi:type="ecore:EAttribute" name="name" eType=
      "ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
   <eStructuralFeatures xsi:type="ecore:EReference" name="artists"
      lowerBound="1"
       upperBound="-1" eType="#//Artist" containment="true"
          resolveProxies="false"/>
 </eClassifiers>
</ecore:EPackage>
```

Creation of a model

If we now want to create the model to be used in our real system, we will have to follow the meta-model created, shown in Figure 3.3. The EMF provides the proper tools for easily creating a model out of a meta-model. By following the same example that we have been discussing, we can create a small final model representing a CD with songs from different artists. The following lines show the representation of a CD with three songs by two different artists.

```
<?xml version="1.0" encoding="UTF-8"?>
<pcd:Artist xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
   xmlns:pcd="http://www.thesis.com/PrimerCD" name="CDMix">
 <songs name="Frozen" originalAlbumName="Ray of Light">
   <artists name="Madonna" numberSongs="1"/>
 </songs>
 <songs name="Billie Jean" originalAlbumName="Thriller">
   <artists name="Michael Jackson" numberSongs="2"/>
```

```
 </songs>
 <songs name="Smooth Criminal" originalAlbumName="Bad">
   <artists name="Michael Jackson" numberSongs="2"/>
 </songs>
</pcd:Artist>
```

As it has been shown with this example, creating the representation of a CD to be used in a real system is very simple. This created model can now be exported to any application, where both the meta-model and the model are going to be used for processing the information. For example, this model could be used in a software application where the real tracks of this CD will be reproduced.

In the context of this work, it is worth mentioning that this example should be understood as a preview for the incoming Chapter 4. One of the principal purposes of the EFIPSANS project is to develop a GANA Meta-Model for the creation of Autonomic Behaviours and its Control Loops, which are obviously going to be modelled based on the created meta-model. The modelled systems together with the Ecore GANA Meta-Model are going to be processed in the EMF environment for achieving some application characteristics.

3.3 The Model-Driven Methodology

Designing the elements involved in an autonomic network by following some software engineering techniques is not a straight-forward task. Thus, when modelling the GANA architecture, we need to describe the functional and logical structures of the elements (DEs and MEs) and their behaviours. To accomplish such a task, the requirements needed for modelling autonomic networks must be analysed and specified [24]. All the requirements analyzed during the EFIPSANS project are listed and described in [19], where the authors provide a deep analysis of the requirements needed for applying a Model-Driven Methodology in the design of GANA elements.

3.3.1 Requirements of the Model-Driven Methodology

EFIPSANS has aimed to develop a *Model Driven Methodology for the design, model-checking and verification of DEs and the simulation, validation and some*

partial Code-generation of autonomic behaviours of DEs. For that reason, the authors of [19], described the needed requirements as:

- *Behavioural Modelling Requirements*: modelling the behaviour of the hierarchical control loops and the elements is needed. This should be done by avoiding some potential conflicts between behavioural characteristics. Thus, some techniques for modelling the behaviour, such as FSM or MSC, need to be incorporated in the methodology.
- *Structural Modelling Requirements*: the structure of the GANA architecture and its entities should be formally specified. This is done thanks to the carefully designed GANA Meta-Model to be used as a designing guidance.
- *Stability Modelling Requirements*: it is intended to approach some of the identified threads compromising the stability of the network by using methods for analysing them. After these analyses it is expected to detect, for example, potential time-scale issues that may cause conflicts between the decisions made by the control loops.

In addition to these requirements, there are also some requirements of the necessary tools to be added in the methodology, which would allow the orchestration of the modelling processes. There are some applications that offer good services for some of the above mentioned requirements; however, they are limited on certain tasks which other tools can offer. For this reason, several tools should be evaluated and analyzed, trying to get from them the best contribution to the whole modelling process, at the same time as fulfilling all the needed requirements. In Figure 3.4 the different steps of the proposed Model-Driven Methodology to be applied for the design, verification, validation and simulation of autonomic entities and hierarchical control loops in a GANA autonomic network are shown.

In Figure 3.4 the logical order to be followed during the design period is also represented. The process starts from the analysis of the needed requirements and it continues with the meta-modelling and structural modelling of the entities forming the network. Once the needed structural characteristics of the architecture are drawn, the modelling of the behaviour is to be done next. After modelling the behaviour of our system, we need to evaluate the design and

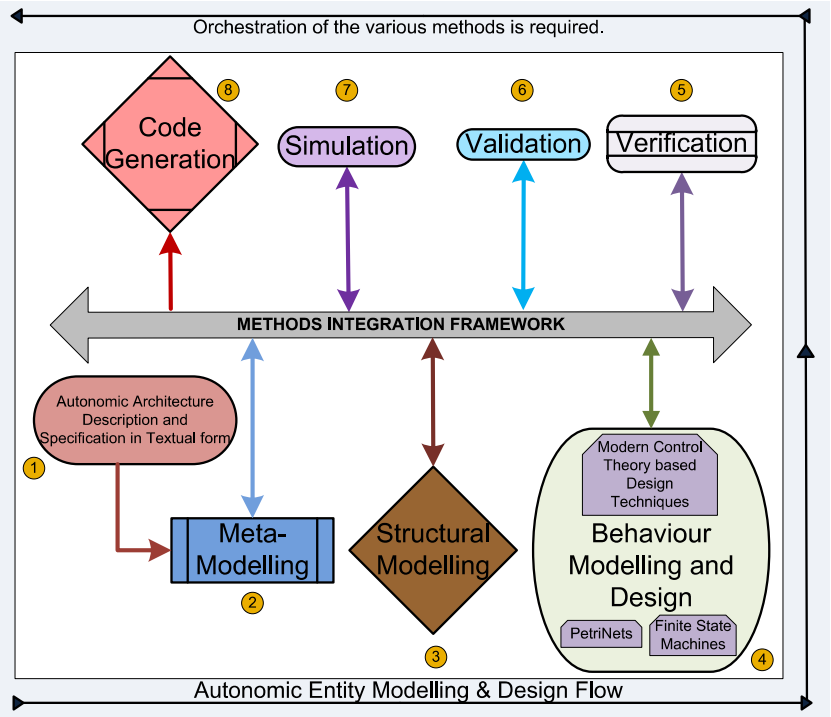

Figure 3.4: Proposed Model-Driven Methodology [24]

check if some of the known stability threads, such as deadlocks, are overcome. For that purpose, a tool is needed that would allow the verification, validation and simulation of the designed system.

In order to ease the job of engineers designing Autonomic Behaviour, several tools are intended to be used under a semi-automated process [24]. To create this semi-automated process the tools need to be integrated within a Methods Integration Framework. This framework aims to provide a fluid orchestration of the different methods offered by the tools, presenting the collection of tools as a unique tool-chain.

All the considered tools, which are explained in Chapter 4, will be working with the same created models. Although different methods will be applied to these models, they all share the same modelling perspective. This interoperation between tools and models is done in the Methods Integration Framework, which works as the central controller of the processes.

Structural Modelling

In order to design the structure of the elements of an autonomic network, it is required that all the networking characteristics associated with the entities and the hierarchical control loops are taken into consideration [24]. These networking characteristics are, for example, the interfaces, the relationship of the control loops, the data types, the communication channels, the information suppliers, etc [19].

Behavioural Modelling

Different processes need to be involved in the behaviour modelling of the entities. The behaviour modelling of the hierarchical control loops is not a straight forward task, as the different natures of and relationships between Protocol, Function, Node and Network control loop levels should be taken into consideration. To cover the different requirements, a hybrid approach for modelling the behaviour has been evaluated. This hybrid approach is based on applying different techniques to the Function-Level control loops and to the upper control loops (Node-Level and Network-Level) [19].

Protocol-Level DEs are those DEs intrinsic to the protocol itself, meaning that the protocol specification has some kind of internal control loop monitoring its own characteristics. This implies that not all the protocols will have an internal Protocol-Level-DE; however, the protocol and its characteristics will continue to be managed by the Function-Level-DEs. Some hierarchical control loops between Function-Level-DEs and Protocol-Level-MEs are modelled applying control-theory techniques. On the other hand, in other cases these hierarchical control loops can be modelled using simpler modelling techniques, such as Finite State Machines (FSM) or Message Sequence Charts (MSC) [24]. The control-theory techniques to be applied in the lowest level control loops are not covered within this thesis. Instead, FSMs and MSCs will be used to model that relationship.

The behaviour of upper level control loops, those running at Node-Level and Network-Level, will be modelled using FSMs and/or MSCs [24].

Stability Modelling

Facing instability in the autonomic networks is one of the biggest threats of this kind for architectures. Several approaches for addressing stability issues have already been mentioned within this document; however, it is important to gather and scope them into the stability subject area.

Different self-* functionalities make parallel decisions that change some parameters in the entities of the network in order to reach a common goal. These parallel decisions might conflict with one another, and thus cause some instability in the network. It is through a tight collaboration between all these self-functionalities that we aim to attain network stability, which is considered the equilibrium point reached and maintained during the existence of the autonomic network. Modelling some stability aspects requires design-period and runtime solutions, achieved through structural and behavioural approaches [25].

The proposed structural solutions and their way of addressing some design-period aspects for reaching stability are:

- *Hierarchical Control loops*: Hierarchical divisions create divided decision processes at different abstract levels. In these divisions the stability problems of node and network levels can be easily differentiated and decoupled. The different levels can be addressed as independent systems working at different timescales.
- *Concept of "Ownership"*: This concept was introduced in Section 2.2.3, and it means that any ME can be just managed by a single DE at any given point of time. This situation avoids that two or more DEs with different decisions can trigger the changes into the ME at the same time.
- *Separation of "Operational Regions"*: The explained GANA control loops are separated based on the nature of the objectives and goals to reach. This will help creating control loops that independently manage different inputs creating divided conceptual outputs. This separation is achieved by creating, for example, different Function-Level-DEs for routing or QoS managements.

In addition to these structural solutions, some behaviour modelling solutions help address the stability issues during the design-period, such as applying well known control-theory techniques that address the stability problems of the systems [25]. Furthermore, stability problems may appear in a running system creating, for example, deadlocks; thus some runtime solutions should address these kinds of stability issues. One of these runtime time solutions is based on the usage of synchronization communication between distributed DEs. By using synchronization means it is intended to addressed conflict resolutions and time-scale issues caused by conflicting decisions to be pushed into the subordinated MEs at the same or similar time [25].

3.4 GANA Meta-Model

As it has been previously mentioned, meta-models are created for representing the systems and act as a blueprint for the engineers to follow. Thus, good quality meta-models should be drawn prior the design of the systems. The meta-model of the GANA architecture is called as GANA Meta-Model, and it has been in continuous development during the whole EFIPSANS project.

The GANA Meta-Model contains the modelling representation of the DEs, MEs and the control loops that govern them [24]. Therefore, the logical and functional characteristics of the architectural structure, together with the behaviour functionalities of the entities building up an autonomic network, are modelled. The design of this meta-model corresponds to the first step of the described Model-Driven Methodology.

The GANA Meta-Model is represented following the UML diagram language. The Generic Modelling Environment (GME) application, which is further discussed at Section 4.1.2, has been used for the creation of the GANA Meta-Model within the EFIPSANS project. GME is an alternative modelling framework to EMF, and even though it provides flexible possibilities for the design of GANA models by using its meta-model, it does not allow the dynamic management of them. Therefore, the EMF framework is used for the management of the created models. These two characteristics involve the necessity of transforming the models created in GME framework to models of EMF kind, this is further explained in Section 4.2.1.

In the EFIPSANS deliverable [24] the whole GANA Meta-Model is deeply explained, however, for the context of this work just few of the meta-model parts are needed. Therefore, in the following subsections I provide an explanation of just those necessary ones. The list of the different GANA Meta-Model parts is:

- GANA Data Types
- GANA Control loop
- GANA Functional Planes
- Dynamic Behaviour
- Events and Actions
- Behaviour Paradigms
- Policy based Control
- Semantic Arc
- Finite State Machine and Coloured Petri Net
- GANA Network Profiles

The parts that needed to be analyzed for the proper evolution of this work are: GANA Control loop, the Decision Plane of the Functional Planes and the Finite State Machines for the behavioural modelling.

3.4.1 GANA Control loop

The concept of control loop was explained in Section 2.2.1. The control loops are the functional loops that will determine the behaviour that DEs need to establish in their subordinate MEs. Thus, for modelling the control loops, the DE-ME relationship has to be specified and the interfaces that govern the communication defined. The needed interfaces were already shown in Figures 2.4 and 2.3, and the meta-model part defining the control loops is shown in Figure 3.5.

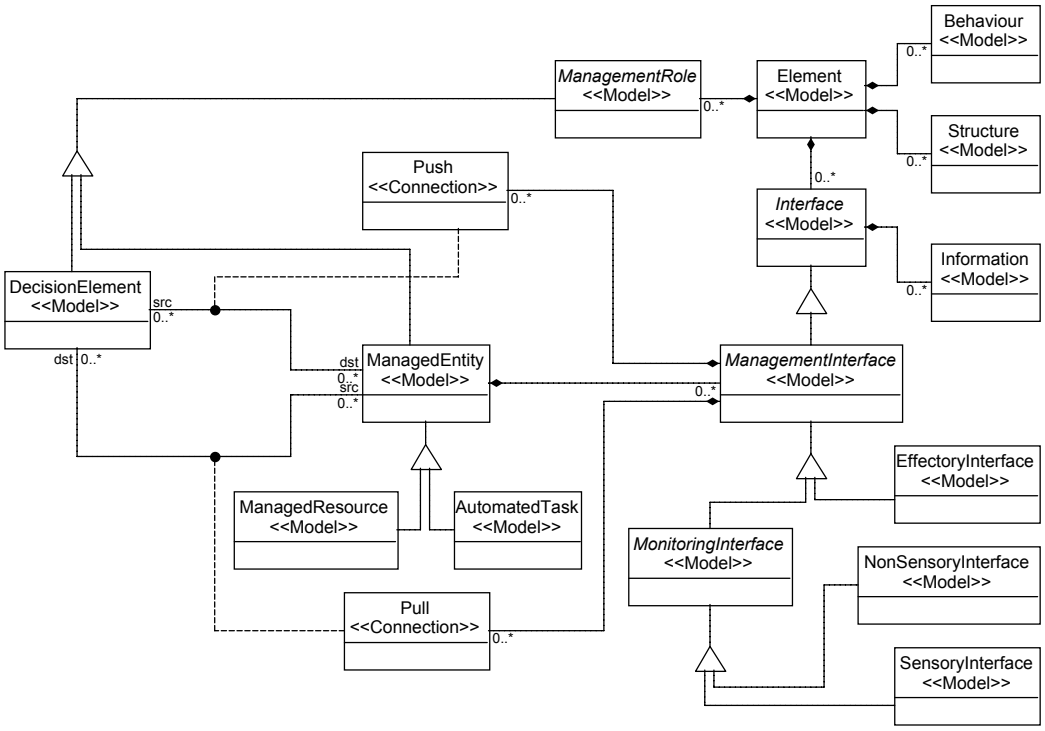

Figure 3.5: Control loop in GANA Meta-Model [24]

The most important parts, among all of them drawn in Figure 3.5, are the Decision Element and the Managed Elements, which are connected by two "Connections" or "ManagementInterfaces": one corresponding to a *pull* communication for retrieving information and one *push* for introducing information. Contained within the "Management Interfaces" are the "Effectors", "Non-Sensor" and "Sensor" interfaces described in Section 2.2.3. Each DE has a "Management Role" depending in the functional abstract level that this DE has been characterised with. In addition, the abstract part named as "Element" is the one containing both structural and behavioural models requested from the requirements of the meta-model [24].

3.4.2 Decision Plane of the Functional Planes

The Decision Plane in the GANA architecture takes care of making the decisions that would drive the behaviours of the elements to achieve a common goal.

The DEs are the entities responsible for triggering the needed decision/changes in the context of the hierarchical control loops. In Figure 3.6 the elements of the meta-model that represent the conceptual parts of the Decision Plane of the GANA architecture are shown.

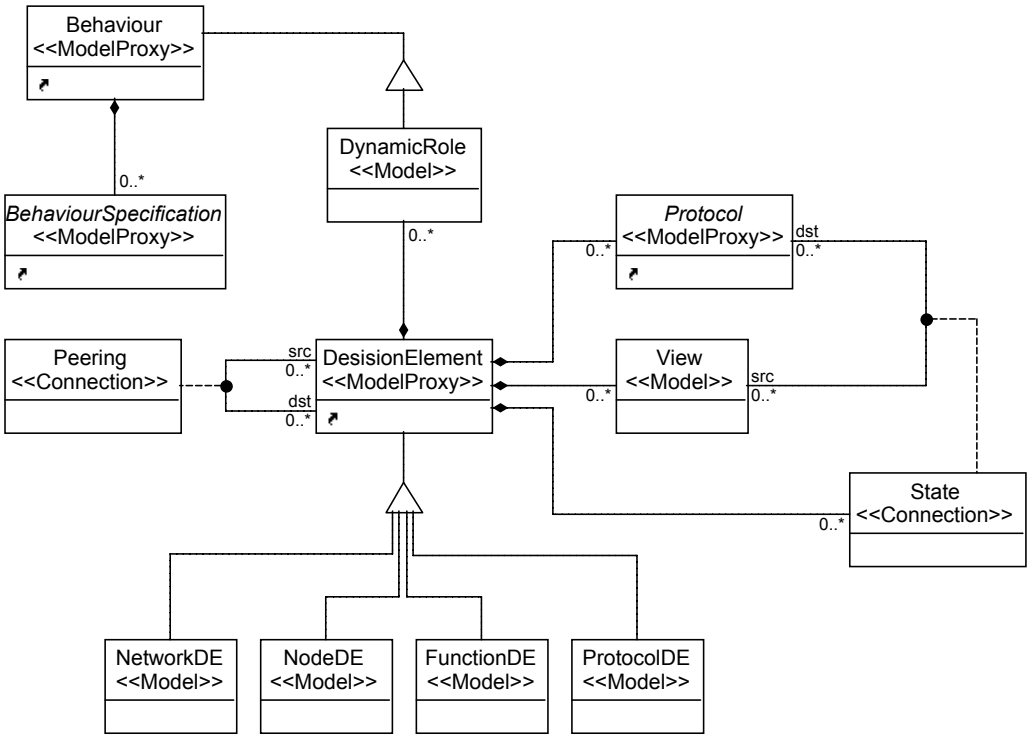

Figure 3.6: Design of the Decision Plane in GANA Meta-Model [24]

As it can be seen in Figure 3.6, the four different "DecisionElements" are described: NetworkDE, NodeDE, FunctionDE and ProtocolDE; which can have assigned different dynamic roles depending in the working level of abstraction and their designed behaviour.

The abstracted model "Protocol", connected to a "View" model, indirectly assigns the characteristics of a protocol that each of the DEs should know for taking the correct decisions that will govern the behaviour of that protocol. In other words, the characteristics that are not relevant to certain level DEs, are hidden from their responsibilities view.

3.4.3 Finite State Machines

Finite State Machine (FSM) is a technique for describing the behaviour of a system based on states and transitions. Mainly two different kinds of FSM are known: Moore FSM and Mealy FSM. In a FSM each state represents a status of the system, such as starting the execution of an action, being in idle mode or had finished the process being described. The transitions between states are triggered when a change in the system is done [26]. These changes can be considered the appearance of an event in the system; such as receiving a message, sending out a message, finishing the execution of a process, etc [24]. The collection of states and transitions building up a FSM is called "Automaton", and the meta-model definition of this abstract concept is shown in Figure 3.7.

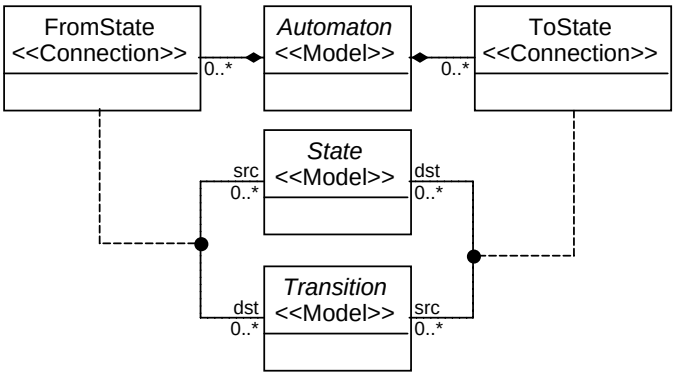

Figure 3.7: State-Transition concept in the GANA Meta-Model [24]

There are clear differences between both kinds of FSM (Moore and Mealy), above all in the concepts that the states and transitions involve. There are two different concepts that determine the behaviour and nature of a FSM: condition and effect.

In the Moore FSMs the states are the effect output of a transition, meaning that a certain transition has led the system being at a specified status or starting the execution of an action. The transitions are conditioned by an event received as input when being at certain state. This input event will trigger the change from one state to the next one, where another output will appear. Figure 3.8 represents a simple Moore FSM. In the Mealy FSM the transitions represent both the input that triggers the action and the output action itself, while the

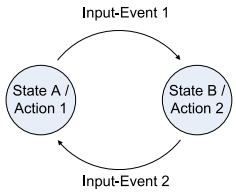

Figure 3.8: Moore FSM

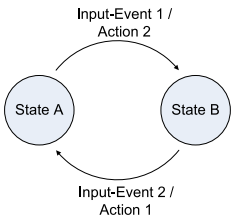

Figure 3.9: Mealy FSM

states represent the result of the action done [26]. In Figure 3.9 the same simple example in Mealy FSM style is shown.

Both types of FSM can be used in the GANA architecture for representing the control loop behaviours. It is worth to remember that the FSM are normally used to model the behaviour of the Node-Level and Network-Level control loops, while mathematical models are used to model the Function-Level control loops. However, there can be situations that mathematical modelling for the Function-Level control loops is not possible or unnecessary; in these cases FSM can be also used for modelling their behaviour.

In Figure 3.10 the meta-model characteristics of both kinds of FSM to be represented are shown. The meta-model represents the two different "Automatons" that can be applied and the different states and transitions that those automatons have. The "Automaton" represents the whole behaviour modelled, and thus it is required that one state is considered the *initial state* where the system starts, and at least one *final state* for representing the accomplishment of the system; although several *final states* can be included in one "Automaton". Between the *initial state* and the *final states* there can be as many *normal states* as needed.

3.5 Summary

In this chapter we have introduced the Model-Driven Methodology to be applied in the design of Autonomic Behaviours. We discussed the structure, behaviour and stability requirements needed for the GANA architecture, and how to model those aspects.

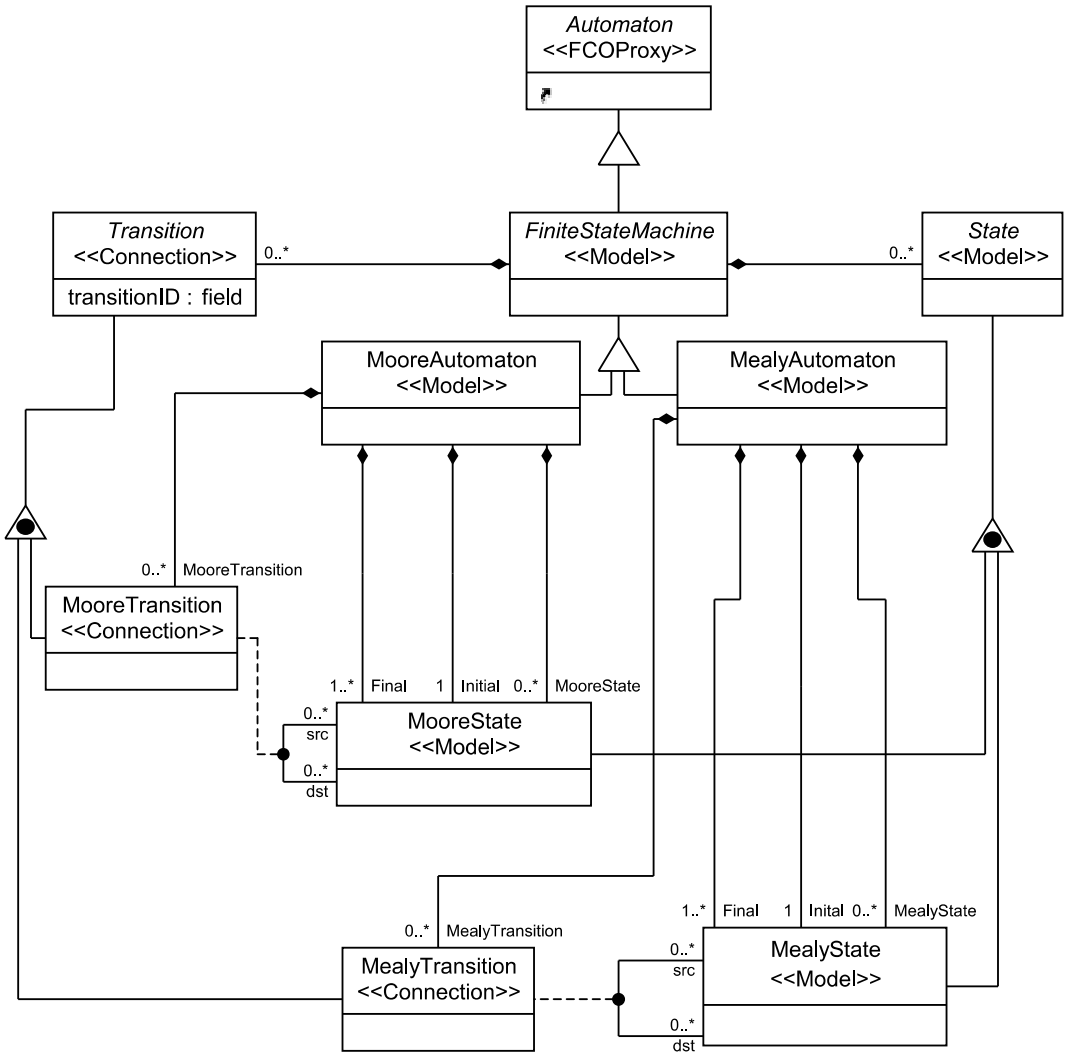

Figure 3.10: Finite State Machine in GANA Meta-Model

The GANA Meta-Model will be used for creating models representing the systems/entities involved in an Autonomic Behaviour, therefore it is important to know the structure and behaviour (FSM) characteristics introduced into the meta-model that will be driving the design of the network entities.

The models representing our system will be described in the Ecore language, which is the meta-meta-model of the EMF framework. EMF is a framework that will allow us to create processes that will dynamically create, delete or modify parts of a model or the entire models.

Chapter 4

Tool Chain

As its name indicates, a tool-chain is a chain of tools, i.e. a collection of tools offering different services that are gathered together creating a tool framework. The primary purpose of a tool-chain is to create an infrastructure offering functionalities from very different tools. The "chain" concept is referred to the guided operability of the tools, meaning that the tools are used in a chain basis as the output of one tool is used as the input of another. One of the biggest problems when facing the creation of a tool-chain is the different natures of the tools, which are built with diverse technologies and normally are not interoperable between them.

The EFIPSANS project has planned the creation of a tool-chain, which based on the modelling requirements would provide the needed services for managing the models and designing real use case scenarios of Autonomic Behaviours. In this Chapter we introduce the different tools of the tool-chain and the services identified to offer some solution to any of the requirements needed for the GANA architecture. In addition, we explain how the framework integrating all the tools is created for achieving the wished interoperability. During the discussion of the tools a small modelling example will be followed in order to help understand the management of the models, however, this example should not be taken as a representation of a real Autonomic Behaviour. A real case study is conducted in Chapter 5.

4.1 Tools of the Tool-Chain

In Figure 4.1 the architecture of the tool-chain defined within the EFIPSANS project is presented. This tool framework has the purpose of supporting the needed modelling steps for the development of GANA Autonomic Behaviours. Each of the included tools offers very specific services to the framework based on their own methods.

The tools of the tool-chain can be divided in three basic groups: the ones offering structural modelling, such as GME [23]; the ones offering behavioural modelling, such as M2Code [31]; and the Methods Integration Framework tool, known as ModelBus [27], which takes the role of central model exchanger. In the following sections the different tools drawn in Figure 4.1 [24] are described.

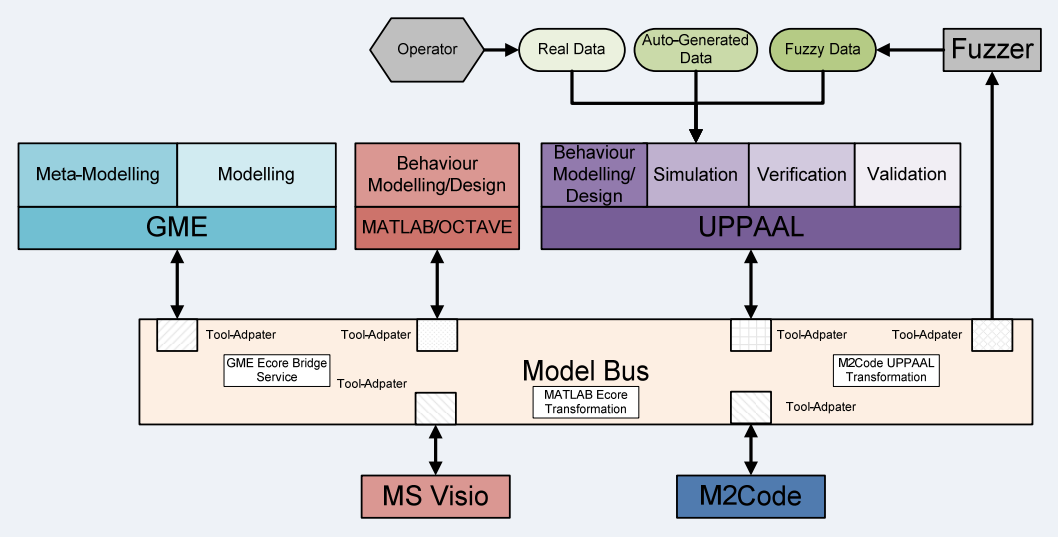

Figure 4.1: Tool-Chain Architecture [24]

4.1.1 ModelBus: Methods Integration Framework

In the context of the Model-Driven Methodology, ModelBus [27] is the application acting as the Methods Integration Framework for the rest of the applications; in other words, the backbone application interconnecting the methods of all the applications. According to [24],

> *ModelBus is a model-driven tool integration framework based on SOA principles and built on industry standards such as Web Services, BPMN, BPEL, OMG standards OCL, UML, MOF (EMF) and JMS*

The rest of the applications connect and communicate with the ModelBus for exchanging the models representing the systems, which are stored in an internal repository. In addition to the storage of the models, this application also manages and executes some actions in the models. Through the needed actions, the ModelBus can change, delete, create or update the models according to the requirements of the next tool in the chain. These actions are carried out by tool adapters.

Tool adapters are needed for each single tool that is intended to be integrated into the ModelBus framework. The tool adapters in ModelBus are responsible for transforming the files created by the external tools into internal understandable data. Although the ModelBus application offers some readily available tool adapters for the most commonly used applications, most of the adapters need to be coded individually. Once a tool adapter is able to transform the created files into means of the ModelBus language, all services offered by ModelBus are available for that tool. Moreover, some additional services or model transformations can be developed and integrated into the ModelBus functionality framework, becoming a service to be used in the context of the whole tool-chain [28].

Among the default services that this application can offer for the integration of different applications, two of them are the relevant ones used in the EFIPSANS tool-chain: Notification Service and Orchestration [28]. The *Notification Service* is the service responsible of the notification exchange between all the services. Notifications can be used, for example, for updating models. For instance, if an application is using a model shared with another application at the same time, the *Notification Service* can be used for notifying the first application when the second one has completed a change in the model that would affect the model being in use by the first one.

The *Orchestration Service* is considered the internal service that allows the execution of all the services in an automatic manner. This helps to create a fluid communication of the services and their capabilities, by using, for example, the

notification service. The orchestration of the services is offered in ModelBus by using web service mechanisms.

ModelBus is a server style application offering a repository. The rest of the applications will communicate with the repository of the ModelBus through proper interfaces.

Generic ModelBus Adapter

The Generic ModelBus Adapter is a client application that acts as the interface towards the ModelBus server. This generic client eases the communication between the tools of the tool-chain and the ModelBus server, as it creates a graphical browser of the ModelBus repository. By using this graphical interface, the rest of the applications can check-in (push) and check-out (pull) models or any kind of file to and from the repository, whose structure is folder based. Further services will internally run at the ModelBus server when the models are checked-in and checked-out, services such as tool-adapter transformations.

4.1.2 Generic Modelling Environment

The Generic Modelling Environment (GME) [23] is a modelling framework acting as an alternative to EMF, which meta-meta-model, based also on UML class diagrams, is known as MetaGME. This application has been selected for the creation of the GANA Meta-Model and the creation of structural models due to the nature of the application, which allows the easy creation of models in a very effective way. The graphical editor of the application allows dragging and dropping elements defined in the meta-model for designing the structure of our systems. In addition, it also checks if the characteristics selected while creating the models conform to the defined meta-model. For example, if a Network-Level-DE is dropped into the graphical working environment, GME will check that no direct connection is done to a Protocol-Level-DE.

The GANA Meta-Model parts shown in Figure 3.5 (Control loop) and Figure 3.6 (Decision Plane) are examples of meta-models created using GME. Actually, the whole GANA Meta-Model has been created using the GME application. We can start the creation of GANA models once the whole GANA Meta-Model is

designed. This is accomplished in GME by indicating in a new project that the reference meta-model to be used in the creation of the models must be the GANA Meta-Model.

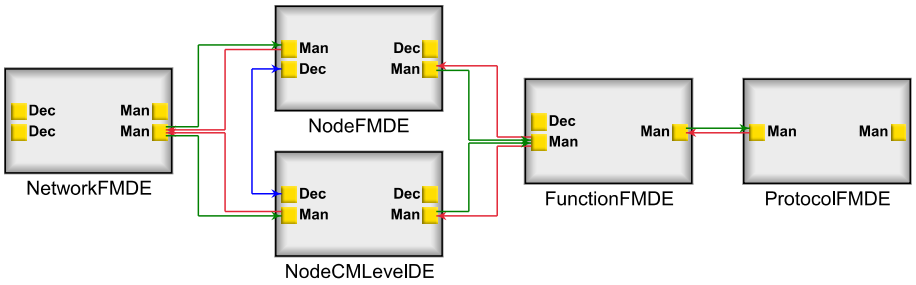

Figure 4.2: Structural model of an Autonomic Behaviour created in GME using the GANA Meta-Model

In Figure 4.2 the structure of a simple GANA Autonomic Behaviour created in GME is shown. This non-representative example shows five DEs: Network, Node, Function and Protocol level DEs for the Fault-Management (FM) and Node level DE for the Configuration-Management (CM). The two Node-Level-DEs are managed by the Network-Level-DE and between them there is a peering relationship (blue line). Each of these boxes includes the needed "interfaces" for communicating with the other DEs. The green connections represent the "forward" communication, i.e. the management interface from upper level DE to lower level DE. The red lines represent the "feedback" communications, i.e. the answer interface from lower to upper level DEs. This is the example that will be used throughout this Chapter helping to understand the modelling process by using the tool-chain.

4.1.3 Microsoft Visio

Microsoft Visio [29] is a diagramming program from Microsoft Corporation used for creating different types of diagrams, such as block diagrams, flowcharts or Message Sequence Charts (MSC). In the EFIPSANS project, and due to other related application limitations, the version included in the tool-chain is Visio 2003.

The contribution that Visio 2003 makes to the tool-chain is based on providing the platform framework where the application M2Code will be run as an Add-in

extension. Within the vast range of services offered by Visio 2003, two of them are relevant for the development of the behaviour models: the creation and modification of user-defined stencils and the usage of drawing sheets. A Visio stencil is a set of diagram shapes gathered together into a sheet, where users are able to see the diagram types that are going to be used in the document. Users will drag these shapes from the stencils and drop them into the drawing sheets, where the design of the desired diagrams is completed. The creation and modification of a user-defined stencil has become a key point in the development of the models conforming to the GANA Meta-Model.

Message Sequence Charts (MSC) diagrams are the models selected for modelling one part of the behaviour of the GANA Autonomic Behaviours. Therefore, Visio will be used for the creation, modification and development of these MSC diagrams. Further discussion about the usage of Visio 2003 is described in the Section 4.1.4.

Connection to ModelBus

For achieving the interconnection between Visio 2003 and ModelBus a COM Add-in has to be installed. The *Visio2003ModelBusAddIn* is an Add-in application that will add two ModelBus buttons into the Visio's toolbars and which will communicate with the ModelBus server, importing/exporting Visio files into/from the repository. The representation achieved after the successful installation of the Add-in is shown in Figure 4.3.

Figure 4.3: ModelBus Import/Export Buttons

These two buttons implement the import and the export functionalities, which will pull/push the models into/from the ModelBus server. Therefore, the ModelBus server must be running to accomplish the selected action. This functionality is implemented by using the *ModelBusClient* application, which is the generic client for communicating between ModelBus and the rest of applications. This generic client interface between ModelBus and the application is further discussed in Section 4.1.1. The Add-in installation file was developed in the EFIPSANS project, however, some constraints at installation time were

found. The Add-in requires to have installed in the machine where Visio 2003 is running the Microsoft Visual Studio 2008 Professional Edition set.

Export of the Model Before exporting the model created, it is recommended to save the file in the local drive. Later by pressing the export button in the toolbar it is possible to check-in the MSC drawing sheet into the ModelBus repository. The Visio 2003 file of the drawing (.vsc) is saved in the selected folder.

Import of the Model If the modelling of the behaviour is processed by only one designer, the drawing/development can be done locally, however, in the case the MSC model is created by different parties is recommended to use the ModelBus repository. Once a model is located at the repository, it can be retrieved into the Visio 2003 environment by pressing the associated import button. This action will launch the *ModelBusClient* window where the selection of the wished Visio drawing file is done. This file is checked-out from the repository appearing at the Visio 2003 environment for later modification.

4.1.4 M2Code

M2Code is an application developed at the Department of Computer Science and Engineering in the University of California, San Diego. This application is implemented by following the definitions and descriptions performed in the Ph.D. Thesis of Mr. Ingolf Krüger "*Distributed System Design with Message Sequence Charts*" [30] and it is widely discussed in the M.Sc. Thesis *Building a Tool for Synthesis of Correct Design from Interaction Specifications* from Mr. Praveen. N. Moorthy [31]. This application is developed to be a MS Visio 2002 and MS Visio 2003 plug-in, thus the inclusion of Visio 2003 in the tool-chain. M2Code is divided in two main parts: the server part and the Visio 2003 utilities. The M2Code server will communicate with the services offered by MS Visio via the COM interfaces. When a MSC diagram is created, the M2Code server will keep track of all the changes performed on the model and therefore being able to later perform the correct MSC to FSM transformation.

The most important service the M2Code application offers to the tool-chain is the creation of Finite State Machines (FSM) models based on the MSC diagram

models. The resulting FSM will be later exported to another tool for further simulation and validation of the model. As described in [31] M2Code is, at high level design, divided into five parts: user interface, model converter, state machine minimiser, layout generators and model exporter. However, not all these parts are relevant for the tool-chain environment. The user interface offered by M2Code is built on Visio 2003 by a user-defined stencil and the related diagram sheets. Because M2Code is a server based solution, which should keep track of the modifications performed in the MSC diagrams, just the shapes offered in the M2Code stencil in MS Visio can be used. The valid M2Code stencil displayed in Visio 2003 is shown in Figure 4.4.

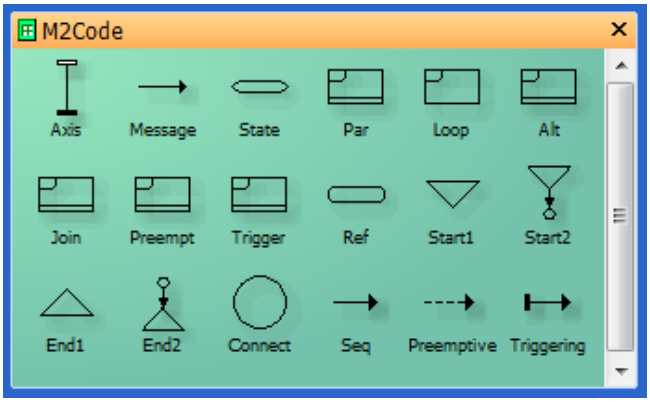

Figure 4.4: M2Code Stencil objects to be used for modelling the MSCs

Deeper description of the different shapes and their usage is found in the M.Sc. Thesis of Praveen. N. Moorthy [31]. From the point of view of the creation of MSC the "Axis" represents the entities and its timeline, the "State" is used to represent marked states, such as the initial or final states, and the "Message" represents the communication between the entities. The model converter part will operate the transformation mechanisms between MSC and FSM, and the model exporter will create the files representing the FSM that will be used to be imported into the ModelBus repository for further transformations.

Connection to ModelBus

As previously described, M2Code is a Visio 2003 plug-in, thus the connection with ModelBus is carried out through the added Visio's ModelBus buttons. Importing the models into M2Code is divided into two separated parts: import

of a module conforming to the GANA Meta-Model, which needs to be used in the M2Code project for modelling the MSCs; and import of an already existing project.

Export of the Model The way to export the M2Code project into the ModelBus is processed by calling the *ModelBusClient* application through the added buttons in Visio 2003. Once the client application launches, the check-in of all the files associated to the current Visio drawing and M2Code project is performed.

Import of Model The limitation in the number of shapes that M2Code allow us to use and the implied requirement based on the fact that the M2Code server must be aware of all the changes performed, has lead us to achieve one suitable solution for imposing into M2Code the structural definition of an Autonomic Behaviour done in GME. This solution is based on modifying the default M2Code stencil to call different Add-in methods that fulfil the GANA model requirements. These methods are created in a Visual Basic for Applications (VBA) module (.bas) to be imported into the M2Code project, further discussion about this is done in Section 4.2. The initially modified stencil should be imported into the M2Code environment in the local machine before an M2Code project is started.

Using M2Code in MS Visio

In the following paragraph the usage of M2Code and its communication with ModelBus are explained in more detail. Continuing with the mentioned example, it will be used for showing the ways of working with the application. At this stage it is assumed that Visio 2003, the ModelBus Add-in and the M2Code plug-in are installed and working at the local machine, and that ModelBus server is up and running.

Import of the GANA model After a GANA model (based on the GANA Meta-Model) is created in GME and checked-in into ModelBus, a ModelBus functionality will be called creating a VBA module (.bas). This transformation service between GME model and VBA module is explained in Section 4.2.2.

Each M2Code project has an associated Visual Basic for Applications project, thus the VBA module created at ModelBus must be imported into the VBA environment of the M2Code project. This module contains three methods that will be called from the modified M2Code stencil, and will check the Visio drawing in such a way that ensures the MSC model conforming to the GANA model designed.

Creation of a MSC project The M2Code server should be launched prior the creation of a MSC diagram in Visio. After that, a M2Code project needs to be created in the MS Visio environment, where we should indicate the project's name and a storage directory in the local machine. In order to start modelling the behaviour of our Autonomic Behaviour the VBA module should be imported into the project's environment. Once this is done, a M2Code-based sheet should be created by calling the macro functionality *M2Code_NewMSC*. This macro functionality will start a communication exchange between MS Visio and the M2Code server, which will be aware of any change performed in the working sheet.

MSC modelling In the created sheet the MSC can be now modelled. By using the shapes in the M2Code stencil we create our diagrams. The "Axis" shape represents the communication entities, in this example the Decision Elements NetworkFMDE, NodeFMDE, NodeCMLevelDE FuntionFMDE and ProtocolFMDE. After drawing and dropping an "Axis" element into the M2Code sheet it is possible to select the names and types that we created in our GANA model, in other words, conforming to the designed model. As example, the available names are shown in Figure 4.5.

In the example, and as shown in Figure 4.6, five components are placed in the drawing sheet. The example created is based on the Network-Level-FM-DE sending a configuration execution order that should be pushed into the Protocol-FM-DE. Because both Node-Level-DE (FM and CM) should be involved in the decision, they exchange peering messages. Later the Function-FM-DE receives the execution orders from the Node-CM-DE and pushes the command into the Protocol-FM-DE. Once the change is done in the Protocol-FM-DE it provides a feedback message to the upper layer DE, which will send an acknowledge message (Ack) for completing the operation. Each DE is at their "initial"

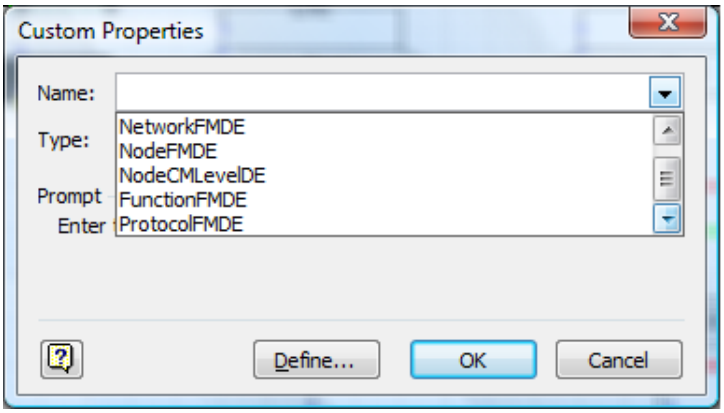

Figure 4.5: Only available DEs names as designed in GME model

status before receiving any message. Once the transaction is completed all the DEs go back to the initial status once again; although it can be considered the "final" stage for this example. At this stage the modelling of the behaviour has been done with MSC. In the Figure 4.6 it can be seen that the peering communication is represented with blue line, the forward with green and the feedback with red as the GME graphical representation shown in Figure 4.2.

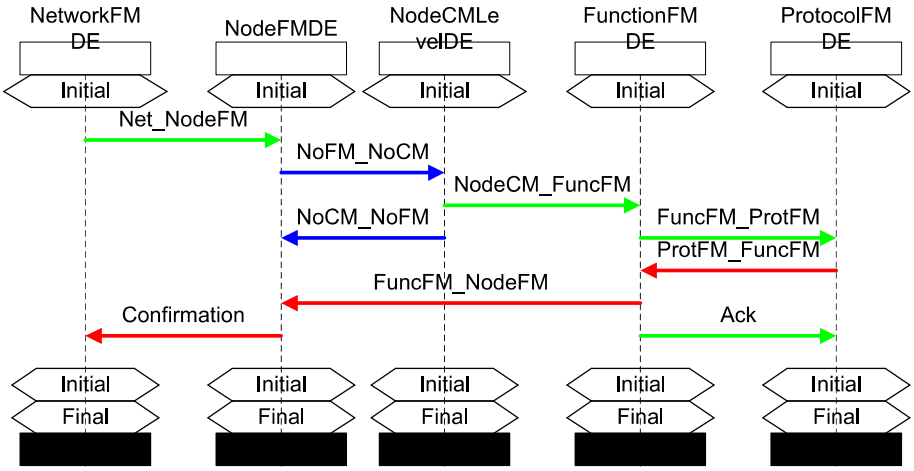

Figure 4.6: MSC representing the modelling behaviour of the Autonomic Behaviour

FSM transformation Once the MSC model is created according to the intended behaviour, the transformation to Finite State Machines (FSM) can be

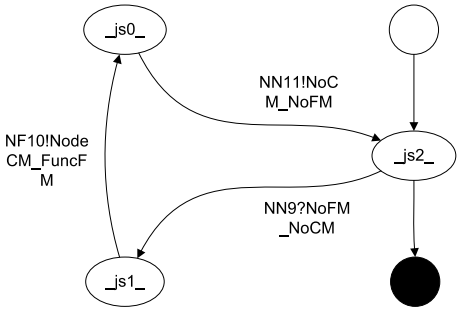

Figure 4.7: Node-CM-DE FSM

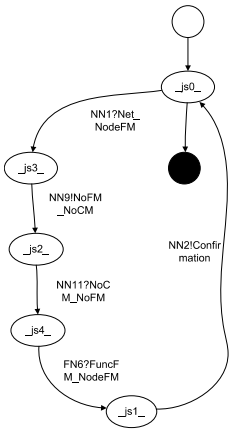

Figure 4.8: Node-FM-DE FSM

started by calling "*M2Code_GenerateStateMachines*", which is a M2Code macro function. Several M2Code windows will pop up allowing us to select the entities for creating the FSMs and their corresponding **initial** and **final** states. In this example the FSM of both Node-Level-DEs is created, as shown in Figures 4.7 and 4.8.

Export of the FSM model After M2Code has created the FSM models its representation a collection of files will be created in the folder selected as project directory. Some of them represent the information of the individual components/entities, while others represent the project as a whole. All of these files should be checked-in into the ModelBus repository for allowing the transformation into the next step of the tool-chain.

4.1.5 UPPAAL

UPPAAL [32] "is an integrated tool environment for modelling, simulation and verification of real-time systems". In this section the tool functionalities and the benefits that UPPAAL can offer to the tool-chain are explained. The contribution that UPPAAL can offer for the design of Autonomic Behaviours is based on the simulation and validation of modelled behaviours in Visio/M2Code.

Importing the model into UPPAAL

UPPAAL is a non-open source Java application; however, in order to accomplish a UPPAAL-ModelBus communication an Eclipse-based transformer was developed. This adapter allows us to open UPPAAL-based models with an UPPAAL editor inside the EMF environment. In addition, this adapter allows launching the UPPAAL tool, which can open the created models for further processes.

In the ModelBus repository all the needed UPPAAL project files representing the models are created based on the information coming from M2Code. Thus, before launching UPPAAL for simulation and validation of our systems, the transformation between FSM to UPPAAL models should be completed.

Using UPPAAL

Once the UPPAAL-based model is opened as a project in the application, the GUI will show the different components/entities and their corresponding FSMs. Continuing with the previous example, it can be seen in Figure 4.9 the representation in UPPAAL of the FSMs belonging to all the DEs. This information was created in M2Code and has been imported into the UPPAAL environment.

Simulation, Validation and Verification

UPPAAL offers two possibilities for the simulation of our system: automatic or manual selection of the FSM transitions. These transitions in UPPAAL correspond to each possible FSM transition belonging to the DEs. The automatic or manual selection of transitions will create a MSC diagram simulating the communication of our systems, as shown in Figure 4.10. The automatic simulation will confirm that our systems are stable, or otherwise detect any deadlock that would have been entered during the design period .

UPPAAL also offers a service for verifying the systems. There are several options available for running the verification according to the necessities for testing our designed models. In [33] a wider system example is explained, together with the associated simulation, validation and verification of the system.

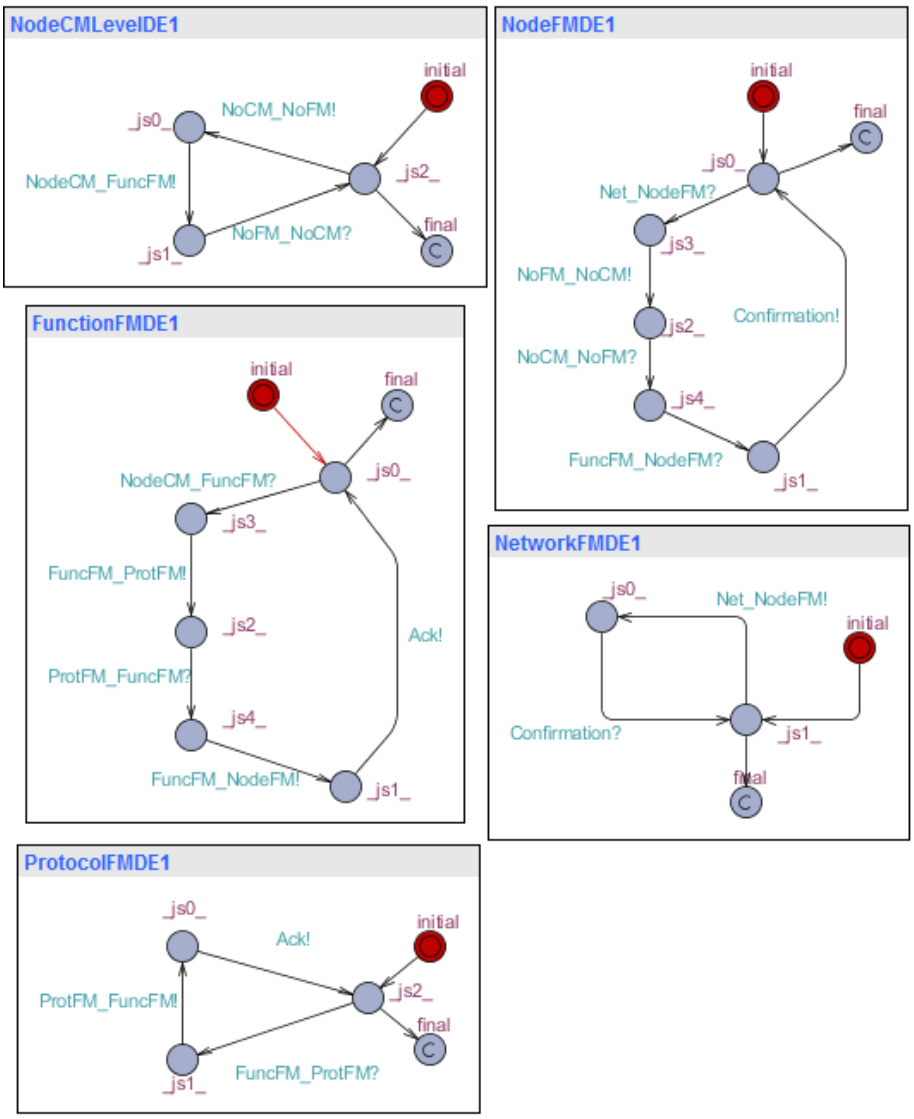

Figure 4.9: FSM of the DEs in UPPAAL

4.1.6 MATLAB-Simulink/Octave

In Section 3.3.1 it was discussed that the behaviour modelling of the Function-Level-DE requires applying some control theory techniques. This is the reason why the application MATLAB Simulink [34] is included into the tool-chain. This application provides the possibility of modelling those kinds of systems requiring control-theory background.

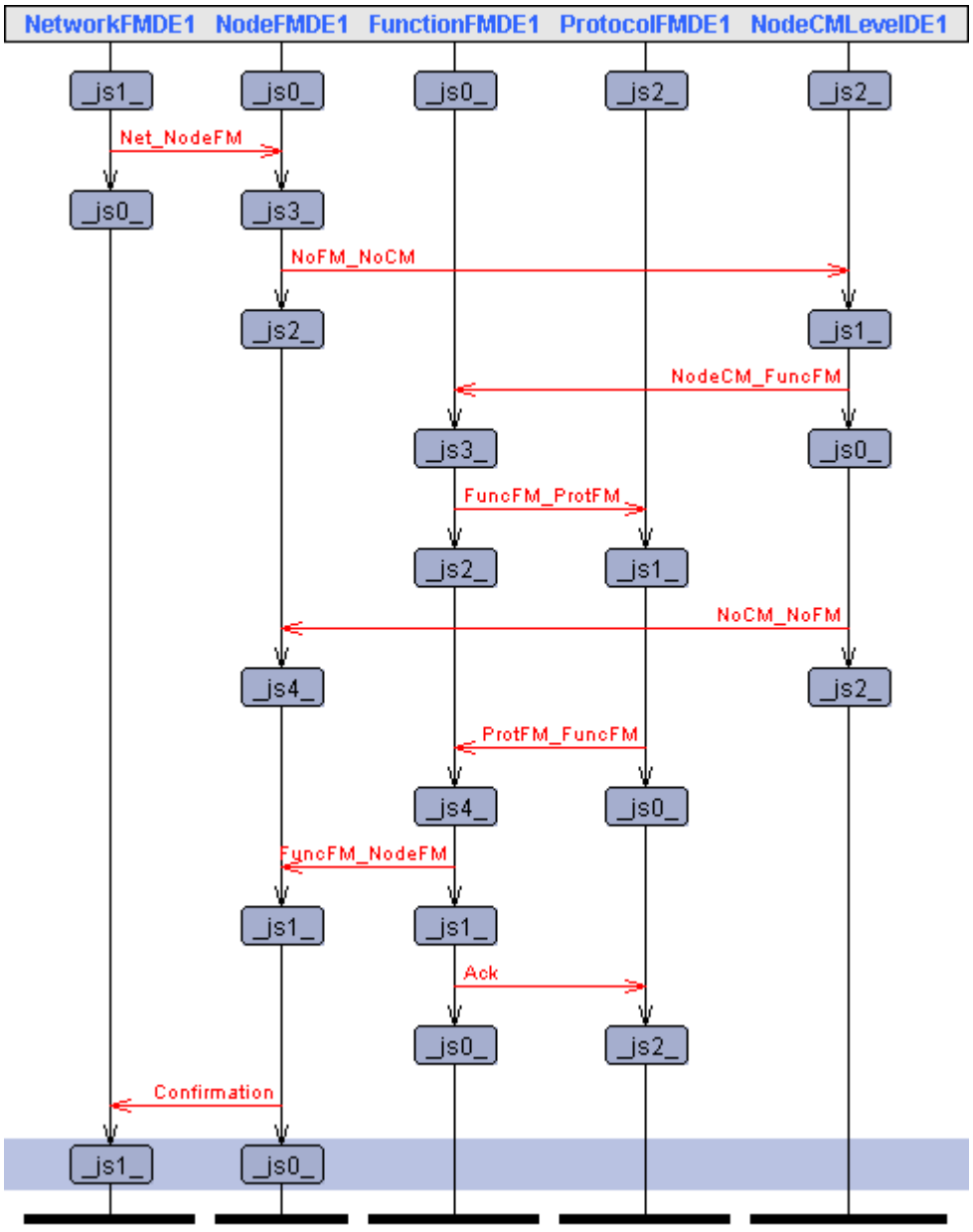

Figure 4.10: MSC of the communication between DEs in UPPAAL

MATLAB Simulink provides the tool-chain with the capacity of modelling the behaviour of the Function-Level-DEs by applying some mathematical methodology to the protocol's behaviour, obtaining as a result a proper controlling DE model of the protocol [24].

4.2 Model Transformations

In the previous sections we have described how different tools can offer a wide range of services for the modelling tasks of the Autonomic Behaviours. Although these tools are built from different technologies, it is still required that they can interoperate and create the feeling in the users of being using a unique tool framework. This capability of representing the tools of the tool-chain as a unique tool framework is achieved thanks to the notification and orchestration services of the ModelBus. The internal technology of the ModelBus for managing the models is based on the MOF modelling standard language, the same language that EMF uses. For that reason, the root language that the rest of the applications should somehow understand is the MOF language. These understanding capabilities are possible by the transformations done at the tool adapters.

From the summary overview of the model flow situation, the first step to be undertaken is the transformation of the GME meta-models and models into the EMF Ecore based meta-models and models. This is carried out by a transformation named *GME-Ecore Bridge*, which has been developed within the EFIP-SANS project. Once the meta-models and models are described in Ecore, the rest of the transformations are done. In the context of this thesis, the model transformation processes needed are sketched in Figure 4.11.

For modelling the behaviour of Function-Level-DEs two alternatives are available. In the first one, the structural part of the Ecore model would be imported into the MATLAB/Simulink Environment, where the behaviour of the Function-Level control loops is modelled using mathematical processes. In the second one, the behaviour of these Function-Level control loops is modelled in M2Code. This latter alternative is the one used in the context of this thesis.

In addition, the M2Code application would be used for modelling the upper level control loops. The transformation from Ecore models into the M2Code environment would include the structure of the control loops, their associated DE elements and the allowed connections between these DEs. Once the MSC model is created in M2Code, the FSMs of each DE are processed. These FSM models need to be transformed into Ecore based language and included into the Ecore model of our system, within the behaviour part of the model. Finally, the

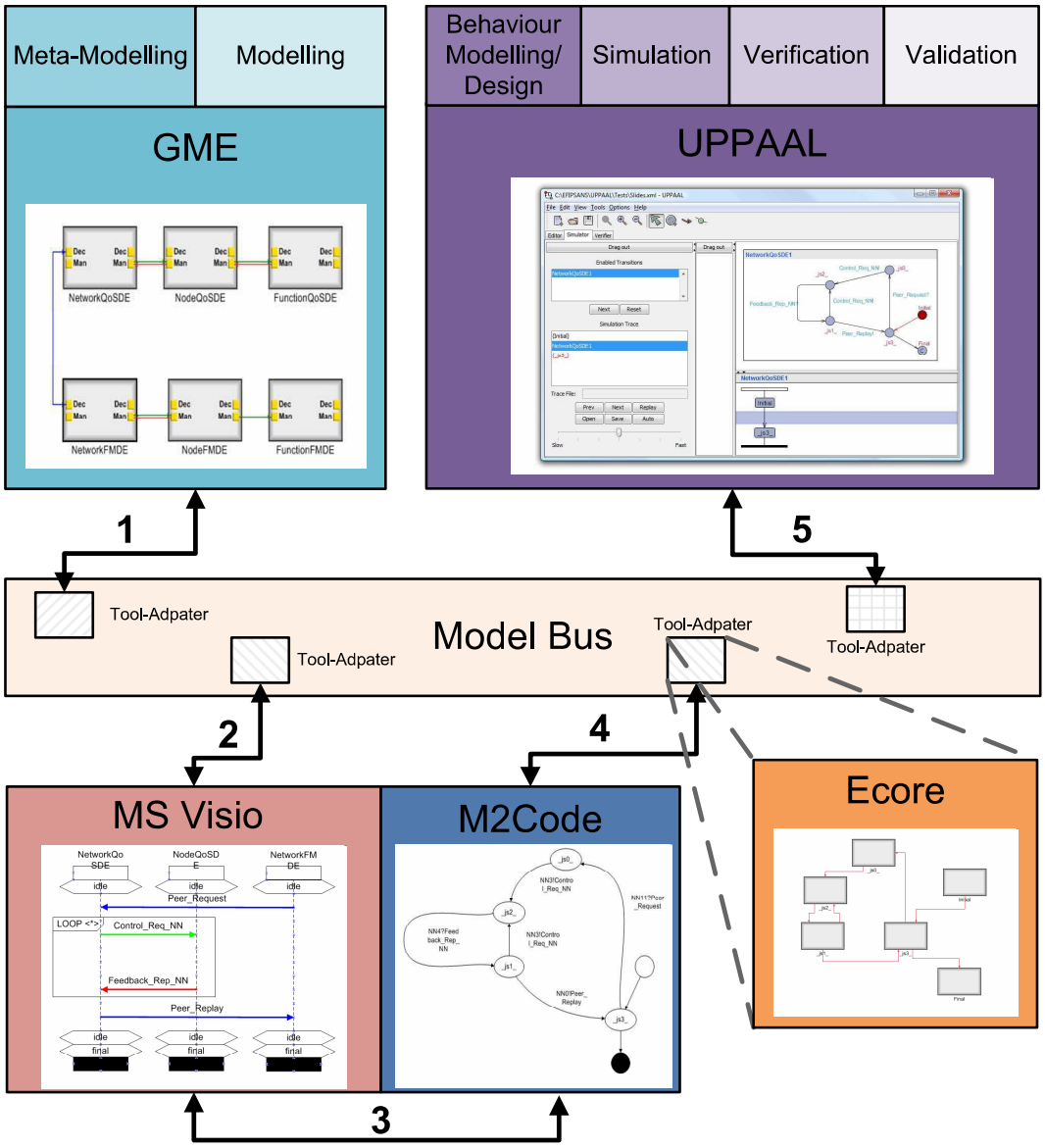

Figure 4.11: Model-Transformation process in the Tool-Chain

FSMs of the Ecore model should be used to create the UPPAAL project file for simulation, validation and verification of our system.

In this section the *GME-Ecore Bridge* and the *Ecore-to-UPPAAL* transformations are briefly introduced, and further analysis is done of the *Ecore-to-M2Code* and *FSM-to-Ecore* transformations.

In the context of this thesis, the collaboration for developing the mentioned transformations have been different. For the *GME-Ecore Bridge* the collaboration was based on the testability of the transformation. For the *Ecore-to-M2Code* and *FSM-to-Ecore* the collaboration was based on the design, development and testing of these transformations, while for the *Ecore-to-UPPAAL* transformation of the collaboration involved the design of the application.

4.2.1 GME-Ecore Bridge

We have already discussed that even through the GME application is used for creating the GANA Meta-Model and the system models of an Autonomic Behaviour, the GME models need to be transformed into EMF-based models before they are processed further. Therefore, a bridge transformation between these two models is needed.

During the EFIPSANS project an already existing approach for transforming between GME and Ecore based models was evaluated. This existing approach is based on three different stages. The first stage is the M3-Level meta-meta-model mapping, where the classes of both meta-meta-models (Ecore and MetaGME) are compared for creating the association map of elements with the same meaning. The second stage is the M2-Level meta-model bridging, where the meta-models of one framework are transformed into the other framework's models by using the map association of stage one. Finally, the stage three is the M1-Level model bridging, where the models are translated by using the meta-model characteristics obtained [24]. This known solution turned out to be unsuitable for the EFIPSANS project as some of the elements that GME allow us to use for meta-modelling are not being transformed/translated into EMF. Thus, an alternative *GME-Ecore Bridge* was developed following this similar 3-stage approach. As commented, part of this thesis work has been based on testing this bridge transformation.

The developed solution offers four transformations: GME $\rightarrow$ Ecore meta-model transformation, GME $\rightarrow$ Ecore model transformation, Ecore $\rightarrow$ GME meta-model transformation and Ecore $\rightarrow$ GME model transformation. For accomplishing this task the bridge uses the GME meta-meta-model (MetaGME) and Ecore meta-meta-model as reference to the rest of the models. Later, through a Java based transformation, the input models from one of the families will be

used for creating the output models of the other family. The application itself uses the EMF framework technology for the control of the models, and it is emended as an automatic service in the ModelBus framework [24].

The reason behind transforming the meta-models and models to Ecore based family is that Ecore models are easily manageable in the EMF environment, and thus Java applications can be used for the dynamic modification of those models to achieve certain goals, such as the creation of specific application files.

4.2.2 Ecore to M2Code transformation

Requirements of the transformation

Let's assume that we have already done the structural model of an Autonomic Behaviour, where different control loop architectures are created. Each of these control loop architectures have different DEs, which are also modelled. If we consider the requirements needed for the M2Code application to model the behaviour of the communication between DEs with MSC, it can be concluded that several structural conditions need to be considered. These requirements are needed as the M2Code environment should conform to both the GANA Meta-Model and the Autonomic Behaviour model created. The requirements are the following:

- Each control loop architecture must be modelled separately and independently of the DEs included on them, in other words, each architecture must contain independent MSC models.
- On each control loop just those DEs that have been included on its structural model can be selected for being included in the independent MSCs. This means that just those DEs with corresponding DE name and DE type are allowed to be included in the model.
- Only those connections between DEs that are included in the GME structural model can be established in the MSC. The M2Code environment should ensure that if a message is sent between DEs which connection is not present in the model, the application will notify the user and forbid that connection.

These requirements suggest that several conditions should be applied in the elements dropped into the M2Code working sheets for controlling the behaviour and structural forms. By checking again the available elements in M2Code, shown in Figure 4.4, it can be concluded that there is a need to control the "Axis" and the "Messages". The "Axis" elements represent the DEs, thus they should be named only with one of the possible range of names and their associated DE types. The "Messages" represent the communication paths between elements, thus the initial and final "Axis" (sender and receiver DEs) should be monitored for prohibiting forbidden connections. In addition, for fulfilling the requirement of being able to model separately the control loop architectures, it is needed to monitor the pages/sheets of the Visio/M2Code environment.

Proposed Solution

Due to the nature of the Visio/M2Code environment, we proposed one suitable solution fulfilling all the requirements. This solution is divided in three different steps: creation of a Visual Basic for Applications (VBA) module with the structural requirements of the GME module, modification of the M2Code stencil properties for calling the methods in the module, and execution of the methods by the elements placed in the working sheet.

By creating the structure of an Autonomic Behaviour in GME we are introducing some characteristics to the system. For example, if we create a system with five DEs with different names and types in GME, as shown in Figure 4.2, we are limiting the range of names and type for those DEs to be between the specified ones. In other words, if we use the model in other application the names and types of the DEs can only be between those specified in GME. Therefore, the representation of those DEs in M2Code should follow this limitation. A similar situation occurs with the communication between DEs. Only those communication connections included in the GME structure can be established between DEs in M2Code.

The inclusion of these limitations/characteristics into the M2Code/Visio environment is done by using the VBA module. The solution found is based on the dynamic modification of the properties of each element in the working sheet.

Each element dragged from the M2Code stencil and dropped into the working sheet of Visio contains a collection of properties defined in a series of tables.

Every master element (Axis, Message, State, etc) of the M2Code stencil contain some default values stored in the properties tables. Thus, when one element is dropped into the working sheet it already contains also these defined default values. On the other hand, once the elements are in the working sheet the default values of the properties tables can be changed, providing different characteristics to each element. In addition, some of the properties not defined in the master element can be defined when they are at the working sheet.

During the evolution of the project, it was considered that a suitable approach would be to modify some of the default properties of the master elements in the M2Code stencil for introducing the structural limitations. These modifications would include calls to VBA functions, which would dynamically modify the properties of the elements in the working sheet, and thus introduce the structural characteristics of the system. Each Visio project has associated a VBA project, thus if a module with functions is added into the VBA environment, the functions can be called from the Visio working sheets.

Because the functions of the VBA module are the ones that will perform the changes in the elements for introducing the structural limitations, the algorithms of the VBA module functions should be created based on the GME models. This transformation between the structural characteristics and the VBA module is done in the EMF framework with a Java application. Thus, the models used are in Ecore basis once the *GME-Ecore Bridge* has transformed them.

Therefore, in order to introduce the structural limitations of the system for modelling the behaviour of an Autonomic Behaviour the stencil of the M2Code should be modified and the VBA module imported into the project.

Modification of the M2Code Stencil

The modification of the stencil is to be done manually. The objective of the modification is simple: introduce in the default properties of the master elements those procedures for calling the methods of the VBA module, so when an element is dragged from the stencil and dropped into the working sheet the VBA module functions are run.

In order to control the names and types to be assigned to the "Axis" elements, which will represent the DEs, a call to the module's method named *DropAxis* is

done. Because the call should be done just after being dropped into the working sheet, the method *RUNMACRO(DropAxis)* must be assigned to the "Event-Drop" property of the master "Axis" in the M2Code stencil. As a result of the call, the properties "Name" and "Type" of the "Custom Properties" table are modified with the only possible selectable names, as shown in Figure 4.5, and the only possible types (e.g. NetworkDE, ProtocolDE, etc).

The functionality for controlling the communications between DEs is different. The requirement of the communication, represented with the "Message" element, is that the two DEs at the end of both connection points are also connected in the GME model. So basically, the method is not called when the "Message" is dropped into the working sheet, but it is rather called when the "Message" is moved and connected to different DEs. This is done with the function *RUNMACRO("CheckMessage")+DEPENDSON(BeginX,EndX)*. This function will check if there is any modification in the "BeginX" or "EndX" positions of the "Message", i.e. if the "Message" is moved within the working sheet. In case there is a modification in either of these two graphical points the module method "CheckMessage" is called. As a result, if the communication between both end-point DEs is allowed, the "Message" will remain, otherwise the "Message" will be disconnected and a pop-up window will alert the user about trying to create a forbidden connection.

Visual Basic for Application Module

As previously introduced, the VBA module will contain the methods that will be called by the elements in the working sheet of M2Code while creating MSC models. A deeper detail of their functionality is provided in this section.

Function "CreateControlLoops" This method is the responsible of separating the different control loop architectures defined in GME. During the modelling of the Autonomic Behaviours in GME, different control loop architectures, with a wide range of DEs, can be designed. Each of these architectures needs to be separated and modelled with different MSCs in the Visio/M2Code environment. This is accomplished by assigning each architecture one different M2Code sheet in the Visio working environment. This function will take care that the correct sheets are created for modelling each control loop separately,

and that their names are according to the information coming from the Ecore model.

Normally this function is called at the beginning of a M2Code project, as it will initially create the M2Code based sheets. However, in case some of the sheets have been deleted, by repeating this function those previously deleted sheets are re-created, leaving the existing ones untouched.

Function "DropAxis" This method is called when an "Axis" is dropped into the working sheet in Visio. In the function there are two variables: (*PossibleAxisAllControlLoops* and *PossibleTypeAllControlLoops*). These two variables contain all the names of the DEs and all their possible types separated by control loop architectures. Thus, when an "Axis" is dropped in the page corresponding to certain control loop, just those DE names and DE types defined in that specific control loop are included in the properties of the DE.

For example, let's imagine the situation where we have modelled two control loops named CL1 and CL2, where CL1 has Net-FM-DE and Node-FM-DE and CL2 has Net-QoS-DE and Node-QoS-DE. At the beginning we would call the *CreateControlLoops* function, which creates two pages in Visio named CL2 and CL1. If we are working in the CL1 page and we drop a DE, the *DropAxis* function will check which DE names and types are included in the CL1, i.e. the names will be Net-FM-DE and Node-FM-DE, and the types Network-Level-DE and Node-Level-DE. The same would happen if we are working in the CL2 page.

The variable with the possible names is *PossibleAxisAllControlLoops*, where the names are stored as "*ControlLoopName1: DE_1_name; DE_2_name; . . .; DE_n_name;*". Similar storage procedure is followed in the variable types. These variables are created accordingly by the Java EMF application, based on the structure of the GME/Ecore model.

Function "CheckMessage" This function is called when any of the "Messages" in the working sheet moves around the page. Its functionality is based on checking each of the end-point connections every time the function is called.

At the beginning, the function will check if the end-points of the "Message" are connected to any DE. In case both end-points ("BeginX" and "EndX") are con-

nected to a DE, then the algorithm continues by retrieving the names of those DEs (sender/source and receiver/destination). Once it has properly retrieved the DE's names it will check with three variables if the combination *DE_begin-DE_end* is included on them. These three variables are *AllowedControlLoop*, *AllowedFeedback* and *AllowedPeering*, and they contain the connections defined in the GME model. In the first one the management interface connections are included, in the second the feedback connections and in the third the peering connections. This entails that in the case two DEs, for example DE1 and DE2, that are connected with the management interface (DE1-DE2) but lack feedback interface (DE2-DE1), the function can properly check if there is a one-way connection or if the connection is in both ways between them.

In the case a "Message" is connected to two DEs, which connection is not included the GME model, the function will show an error window and the connection will be eliminated from the end point, while remaining connected the source one.

Creation of the VBA Module in EMF

The previously described VBA module has three functions that will be called for introducing the structural characteristics in the elements of the Visio working sheet as defined in GME. Therefore, it is required that the algorithms of those three functions are built based on the GME structures defined for each Autonomic Behaviour. The necessary structural characteristics to be included in the algorithms for each Autonomic Behaviour are: the DE names, the DE types, the control loop names and the allowed connections between DEs (forward, feedback and peering). These characteristics are used within the algorithms of the VBA module in the variables: *PossibleAxisAllControlLoops*, *PossibleTypeAllControlLoops*, *AllowedControlLoop*, *AllowedFeedback* and *AllowedPeering*. Therefore, the content of these variables need to be specified based on the information of each Autonomic Behaviour designed.

In order to create different VBA modules representing each specific Autonomic Behaviour designed, a Java application is needed. This Java application will get the structural information from the model and create the variables with the proper information. Finally, once the variables contain the structural information, the VBA module is created.

We have developed this Java application, which is based on the Dynamic EMF technology, within the context of this project. As input of this application we introduce the Ecore model containing the structural information. This Ecore model is actually the output model of the *GME-Ecore Bridge* transformation done in ModelBus. After processing the Ecore model, it creates the VBA module as output of the application. Thus, each Autonomic Behaviour created in GME and checked-in into the ModelBus will have its corresponding VBA module.

The internal functionality of this application is based on retrieving the structural characteristics from each Ecore model. The first part of the application checks all the control loops stored in the Ecore model, obtaining its names and saving them into a table. Later, it processes each of these control loops retrieving every single DE's properties (name and type), and their corresponding allowed communications. This information is also stored on the same table, which at the end contains the structural characteristics of the Autonomic Behaviour separated by control loops. The information in this table is the one used for creating the dynamic variables of the VBA module. Once these variables are created, they are included in an internal VBA module template. At the end of the application, the VBA module template with the dynamic structural information is saved as a VBA module file (.bas) in the ModelBus repository.

This Java application is supposed to become an integrated service in the ModelBus. When the GME meta-model and models are checked-in into the ModelBus repository, a notification is sent to the *GME-Ecore Bridge*. Then, the GME models are used as input of the bridge and the Ecore meta-model and models are the outputs. Once the process of the bridge is completed, a notification is sent to this service, which will use the Ecore meta-model and model as input, creating the VBA module as the output of the procedure.

4.2.3 FSM to Ecore transformation

The M2Code application is used for transforming MSC models into Finite State Machine (FSM) models, which are associated to each DE being modelled. Modelling the behaviour of the DEs with FSM is considered an effective way for those DEs at Node-Level and Network-Level. In addition, FSM can also be used for modelling the lowest two levels, i.e. the Function-Level and the Protocol-Level.

In order to fulfil the orchestration requirements through the whole tool-chain, the information of the FSMs must be included in the Ecore model under the behaviour part of the system's model, as shown in Figure 3.5. This is done by using the Dynamic EMF technology together with the FSM meta-model of the GANA Meta-Model sketched in Figure 3.10.

The application for transforming the FSMs, obtained from M2Code, into the Ecore model is divided in three main tasks: analysis of the files created by M2Code with the FSM information, creation of representative tables and modification of the Ecore model for including the FSM models.

Analysis of the M2Code Files

M2Code creates a whole set of files containing the information of the FSMs. All these files contain relevant information for the transformation to Ecore model, thus the whole set of files must be checked-in into the ModelBus repository, where a parser will be called for analysing theses files.

This parser is a Java application that would first check all the file names for acknowledging the structural characteristics and then go inside the files for obtaining further information about the FSM. For building the structural characteristics the ".m2s" type files are checked. The names of these files are structured as "<DE_name>@<Control loop_name>", what allows separating all the DEs into their corresponding control loop architectures. In addition, these files are used for retrieving the exact FSM states that belong to each particular relation DE@ControlLoop. This is useful for those cases where the same DE is included in more than one control loop architecture.

Once the parser knows about all the DEs available, it checks the ".dot" files, which are named as "<DE_name>.dot". These files contain all the FSM states of the DEs. For example, if we consider the FSM of Figure 4.7 that belong to the Node-CM-Level-DE, the file will be named as "NodeCMLevelDE.dot" and its content is:

```
digraph component {
    edge [arrowhead = none];
    _js0_ -> _js2_ [label = "NN11!NoCM_NoFM"];
    _js1_ -> _js0_ [label = "NF10!NodeCM_FuncFM"];
    _js2_ -> _js1_ [label = "NN9?NoFM_NoCM"];
    _js0_ [label = "_js0_", color = blue];
    _js1_ [label = "_js1_", color = blue];
    _js2_ [label = "_js2_", color = blue];
    0 -> _js2_ ;
    0 [shape = circle , color = red , label = ""];
    _js2_ -> 1 ;
    1 [shape = circle , color = black , label = ""];
}
```

It can be seen that the states, their transition and the corresponding messages are represented very clearly in the file. Even the initial (with a "0") and final (with a "1") states are shown, what helps to build up the needed information about the whole FSM.

Creation of representative tables

During the parsing process, two tables containing the information of the FSMs are created. These two tables are being filled up through the whole parsing process, and are the ones to be used for including the FSM models inside the Ecore models. One of the tables will represent the information of the states of the FSM, while the other will represent the transitions between those states.

The "*controlLoopAndAxis*" table will associate the number of states and their names with each single DE. Furthermore, each DE is separated into the control loop the FSM belongs to. It is possible that one DE is included in more than one control loop within a single Autonomic Behaviour, and thus having different FSMs belonging to the different control loops. By following the example proposed in Section 4.1.2 and which FSM are shown in Figure 4.9, the information obtained in the mentioned table is shown in Table 4.1. This table represents for each DE the control loop name (CL Name), the Decision Element name (DE Name), the number of states (#) and the name of the states

(States). In this example the DEs belong to the control loop architecture named as *TheCL*.

Table 4.1: Table with the FSM States information

CL Name	DE Name	#	States
TheCL	FunctionFMDE	7	Initial;Final;_js0_;_js3_;_js2_;_js4_;_js1_
TheCL	NetworkFMDE	4	Initial;Final;_js1_;_js0_
TheCL	NodeCMLevelDE	5	Initial;Final;_js2_;_js1_;_js0_
TheCL	NodeFMDE	7	Initial;Final;_js0_;_js3_;_js2_;_js4_;_js1_
TheCL	ProtocolFMDE	5	Initial;Final;_js2_;_js1_;_js0_

On the other hand, the table "*transitionTable*" will store the information of the transitions and their corresponding messages by associating the source states and the destination states. If we consider the FSM of Figure 4.7, the information corresponding to the transitions of this DE is represented in Table 4.2. This table represents the control loop name (CL Name), the Decision Element name (DE Name), the source state (Src), the destination state (Dst) and the transition's message (Message).

Table 4.2: Table with the FSM Transitions information

CL Name	DE Name	Src	Dst	Message
TheCL	NodeCMLevelDE	_js2_	_js1_	NN9?NoFM_NoCM
TheCL	NodeCMLevelDE	_js1_	_js0_	NF10!NodeCM_FuncFM
TheCL	NodeCMLevelDE	_js0_	_js2_	NN11!NoCM_NoFM
TheCL	NodeCMLevelDE	initial	_js2_	Initial
TheCL	NodeCMLevelDE	_js2_	final	Final

Including the FSM model into the Ecore model

After the parser has accomplished the creation of the tables, the process for including the FSM information into the Ecore model is initialized. This process is based on a Dynamic EMF transformation, where states and transitions create a whole "Automaton" element, which is introduced within each of the

DEs involved. At the end, each DE in the global Ecore model has a behaviour model conforming to the FSM meta-model of the GANA Meta-Model, which was shown in Figure 3.10.

If the resulting Ecore model would be transformed into GME model using the *GME-Ecore Bridge*, then the FSM information of the "NodeCMLevelDE" would be graphically represented as shown in Figure 4.12.

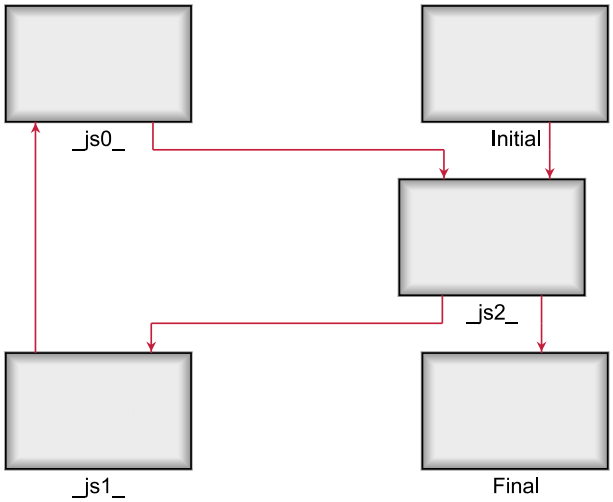

Figure 4.12: Graphical representation in GME of the FSM model of one DE

4.2.4 Ecore to UPPAAL Transformation

The final intention behind modelling the behaviour of the DEs with FSM is that the Autonomic Behaviour can be simulated, validated and verified in the UPPAAL application, which was introduced in Section 4.1.5. Thus, in order to simulate, validate and verify our designed Autonomic Behaviour, a project in UPPAAL is needed. This project should contain the same FSM information corresponding to the DEs as the information obtained from the M2Code application, as the example shown in Figure 4.9.

UPPAAL projects are saved in a single XML format file, what means being human readable. This project file can be created in ModelBus with an internal service and imported into the UPPAAL environment by using the Eclipse-based transformer. Therefore, a process that would take the Ecore model as an input

with the FSM information and produce an UPPAAL XML project file as an output is needed.

However, there is a major drawback in the creation of the UPPAAL project file from the Ecore model. This project file not only represents the information about the FSM (states and transitions), but it also contains the exact graphical representation of those elements in the working sheet of UPPAAL. The Ecore FSM models only contain abstract information, but no graphical data from M2Code. This situation requires complicated mathematical and graphical calculations when creating the UPPAAL XML project, otherwise the application will create a graphical mess of states and transitions that will confuse the user of the application.

For the purpose of achieving the whole tool-chain orchestration, a complex application acting as UPPAAL project builder is required. This application would be divided in three steps: retrieval of Ecore FSM information from the Ecore models, creation of representative tables and completion of an UPPAAL project file template with the needed arguments. Once these steps are successfully accomplished, the UPPAAL application can open the created file as its own project and proceed with the simulation, validation and verification of the system.

4.3 Summary

In this Chapter we have introduced the different tools of the tool-chain and described how the flow of models occurs between different tools by using the services of a centralized framework called ModelBus.

The first tool used for the design of Autonomic Behaviours is GME. In this application the models of the system are created by using the GANA Meta-Model as reference model. These models are introduced into ModelBus, where a transformation called *GME-Ecore Bridge* will translate these GME models into Ecore models.

The Ecore models will be used in another transformation for creating a VBA module to be used on the M2Code/Visio environment. This VBA module will be responsible for checking that the structural characteristics of the system are followed in M2Code. In the M2Code/Visio environment we will define the

behaviour of the DEs with MSCs. Later, using also the M2Code application these MSCs will be transformed into FSMs.

The files created by M2Code with the FSM information will be checked-in into the ModelBus framework, where another transformation will include the FSM information into the global Ecore model representing our Autonomic Behaviour.

A final transformation will create an UPPAAL project file from the FSM information stored in the Ecore model. The UPPAAL application will use the same FSM of each DE created in M2Code for simulate, validate and verify the system.

Chapter 5

Case Study

In this Chapter a case study of the discussed technologies is reviewed. The intention is to showcase the usage of the described Model-Driven Methodology and the Tool-Chain in the design of Autonomic Behaviours.

In the first part of the Chapter a networking scenario is defined, which includes aspects of routing and cooperation between operators in a multi-domain environment under risk management assessments. In the second part of the Chapter the Model-Driven Methodology and the Tool-Chain will be used for designing the Autonomic Behaviour of the proposed scenario. The work includes the structural modelling of the networking scenario, modelling the behaviour of the involved DEs with MSC in M2Code, and finally, the analysis of the designed system in UPPAAL. In this final stage the simulation and validation of the system is to be done. After that, the results will be evaluated and possibly detect design problems.

5.1 Networking Scenario Definition

Several networking scenarios and their associated Autonomic Behaviours have been described within the EFIPSANS project. To provide a standardized format a template was used for the description of those scenarios [17]. To align this work with the EFIPSANS project, the same template is used for describing the scenario. The scenario's description is shown in Table 5.1.

Table 5.1: Description of the Networking Scenario

Auto-Collaboration between Network Providers for Self-Adaptation of Routing as driven by Risk-Level Assessment in a Fixed Network Environment	
The Story-line	Fault management in today's networks is based on reactive actions done by the centralized system responsible for administrating the network. These reactive actions involve processes from the network administrator to locate, isolate and repair the faults. Therefore, there is a lack in proactive risk management actions that would act in real-time for avoiding the appearance of faults in the network. These kind of proactive strategies would successfully avoid some of today's network problems that drastically affect the performance. One of the strategies that would allow some real-time risk management is based on the modification of a routing protocol, such as OSPFv3 [24]. The characteristics of the OSPFv3 protocol would be modified to achieve a dynamic adaptation to the current risk situation. An example of this adaptation could be the change in link weights that limit the traffic in the router when the temperature of a router rises. This situation is aggravated when considering the border gateway routers. These routers interconnect different network providers, thus a failure on one of them would create problems to more than one network and even cause financial consequences, i.e. that the network operator cannot fulfil the agreed SLAs. The high importance of these routers should be noticed, as the failure in one of them could mean a break-down in the performance until the routing protocol readjustment finally converges. In addition, it is a fact that network operators do not want to exchange any information about the status of their network that could carry any kind of business damage. Thus, a solution for auto-collaboration under confidential conditions is needed. This solution can be provided by GANA compliant networks, where the ONIX systems store some data from other operator's networks at an abstract level. The combination of both solutions (auto-collaboration and risk management) can provide higher availability to the network, by lowering the appearance of faults or damages. This process involves concepts of resilience and survivability capabilities introduced in the GANA autonomic networks.

Short description of the scenario	This Autonomic Behaviour scenario tries to show the benefits of auto-collaboration procedures between network operators when a potential high risk fault has been detected in one of the border gateway routers. The auto-collaboration procedure involves self-description and self-advertisement characteristics. The potential fault detected is based on the high temperature reached by one of the routers. The raised risk alarm at one of the network operators triggers the auto-collaboration process whose final purpose is to gradually eliminate the traffic crossing that router. The final purpose of the process is to isolate and repair the faulty router without altering the network traffic and its performance.	
Current problems with current practices or current technology	Today's routing protocols calculate the routing decisions based only on traffic metrics. They do not introduce any kind of risk metrics that would lead to different solutions or dynamic adaptation to them. This leads to not having effective networking frameworks that would allow proactive actions to be taken for avoiding fault appearances. Furthermore, exchanging information that describes current network status between operators does not take place. Each network operator wants to maintain secrecy about their own risks, problems and faults, without considering that they can affect other networks operations.	
Network Environment	Fixed Network	
Self* Funct.	**Self-*Functionality**	**Problems/Limitations it addresses**
	Self-Adaptation	Proactive actions would allow preventing faults or problems to appear in the network. Current solutions only allow reactive actions, which implies a time-frame of instability.
	Self-Description & Self-Advertisement	Today's network operators do not exchange information describing and advertising the devices capabilities due to secrecy limitations.

<table>
<tr><td>Self- Adaptation – What it solves and the benefits</td><td colspan="2">The Self-Adaptation mechanism introduced in GANA aims to include procedures that would allow triggering proactive actions for avoiding potential faults or damages in the network and their devices. This is achieved by changing the routing capabilities of the involved routing protocols, such OSPFv3 and BGPv4, by isolating the devices under high risk avoiding traffic disturbance.
The final purpose of the Self-Adaptation under risk assessments is to increase the capabilities of self-resilience and self-survivability introduced in the GANA architecture.</td></tr>
<tr><td>Self-Descrip. Self-Advertis.</td><td colspan="2">Self-Description and Self-Advertisement allow the networks to be informed about the capabilities of the devices. Through the ONIX system, the Network-Level-DEs can get information about other network operator devices and capabilities. This information retrieval would allow Self-Adaptation to the network conditions.</td></tr>
<tr><td>System(s) Involved</td><td colspan="2">1. Different ISP network operator with Core Routers and Border Routers
2. Content Providers and end users
3. ONIX System
4. NET_LEVEL_R&S_DE (Network-Level-Resilience&Survivability-DE) and NET_LEVEL_RM_DE (Network-Level-RoutingManagement-DE)
5. NODE_MAIN_DE</td></tr>
<tr><td rowspan="2">Key players that benefit</td><td>Player</td><td>Benefits</td></tr>
<tr><td>Operator</td><td>Different network operators will benefit as the interconnection between them increases its availability. This will create a more reliable network, allowing a better relationship between network operators.</td></tr>
</table>

	Manufacturer	By creating GANA compliant devices, the manufactures will be able to offer network operators more reliable devices that will increase the overall network availability.
	End User	End users and content providers will benefit from a more reliable infrastructure. In addition, business relationships between network operators and content providers are increased, as the reliability SLAs can be fulfilled.

The main intention of this case study is to evaluate the applicability of the Model-Driven Methodology and the Tool-Chain in the design of Autonomic Behaviours. However, even a secondary intention is to show the potential possibilities that an Autonomic Behaviour has for achieving certain autonomy in the network, this example should not be taken as an accurate and clear reflection of a real Autonomic Behaviour as described in the EFIPSANS project. The reason is that some requirements might not be entirely fulfilled.

Detailed Description of the Scenario

The Autonomic Behaviour of the introduced scenario has a clear goal: isolate the router where a high risk alarm has been detected, diverting its traffic to alternative possibilities without altering the global network's traffic. For achieving this goal it is intended to modify the routing information entries in the Routing Information Base (RIB) and Forwarding Information Base (FIB) tables that are maintained by the Open Shortest Path First version 3 (OSPFv3) and Border Gateway Protocol version 4 (BGPv4) protocols. These modifications are intended to be done before the protocol's algorithms converge and simultaneously on both sides of the network. In Figure 5.1 is graphically representing the networking scenario to be designed.

Before describing the Autonomic Behaviour process, some assumptions have to be made. It is assumed that both network operators or Internet Service Providers (ISP) have GANA compliant networks and network devices. Both ISPs ONIX systems are up, running and are aware of each other, this means that both Auto-Configuration and Auto-Discovery of the capabilities have been

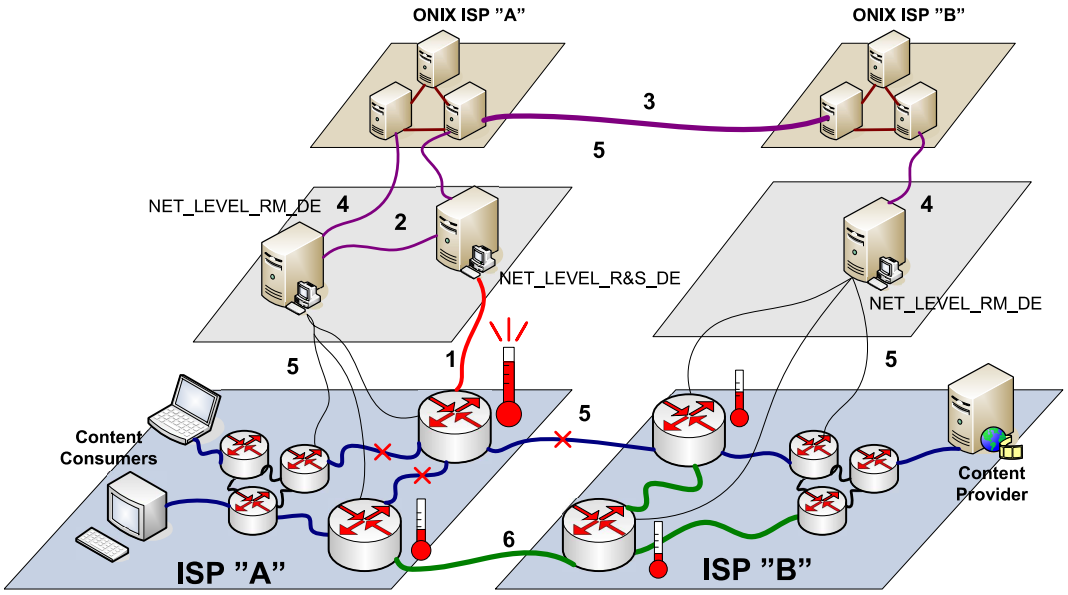

Figure 5.1: Networking scenario for the Case Study

established. Each network element has its NODE_MAIN_DE running, and it is being managed by the needed Network-Level-DEs. In addition, the two Network-Level-DEs (NET_LEVEL_R&S_DE and NET_LEVEL_RM_DE) have been properly set up.

Two different ISPs are involved in this networking scenario. In one ISP, known as ISP "A", there are normal Internet users consuming a service offered by a content provider. This service requires a high exchange of network traffic, above all in the downloading direction, i.e. from the content provider to the end users. The content provider's servers reside with the other ISP, known as ISP "B". Both ISPs have core routers and two border routers that interconnect their respective networks. Initially, the traffic that is being downloaded by the users follows the dark-blue path of Figure 5.1.

Although each network is monitored independently, their ONIX systems are exchanging confidential information for Auto-Collaboration, such as link metric capabilities. Both operators know about the network performance risks that would suggest a sudden fault in any of the border routers, thus they have agreed to exchange information to proactively address this problem. On the other hand, they want to prevent any relevant information from being understood by the other operator, this is why the information to be shared with other

network operators is in an abstract level definition [17]. This information can be stored in the other network operator's ONIX systems, without providing any relevant information about the network's topology or performance characteristics.

The Autonomic Behaviour process starts when ISP "A" is auto-monitoring the status of the network elements and a sudden critical event is recorded after a temperature threshold is reached. This event is sent by one of the border routers, which has reached a high CPU temperature level, likely caused by a cooling system being damaged. The NODE_MAIN_DE of the router informs the Network-Level-Resilience-&-Survivability-DE (NET_LEVEL_R&S_DE) about this event, represented in Figure 5.1 with step "1". The NET_LEVEL_R&S_DE evaluates the situation and realizes that such sudden event carries a high risk that could drive the router to become severely damaged and in turn finally breaking down. This situation would lead the interconnection between ISPs to break and the connection "content provided-end user" to be interrupted. The NET_LEVEL_R&S_DE informs the Network-Level-RoutingManagement-DE (NET_LEVEL_RM_DE) in step "2" about the situation and the necessity of isolating that device.

The NET_LEVEL_RM_DE evaluates the current situation and considers the needed routing changes to be triggered for isolating that router, before it finally breaks down and the connection is totally lost. This DE knows that the device is a border router that requires inter-collaboration with the other ISP's devices for ensuring the reliability of the connection. To achieve this, the Network-Level-DE informs the ONIX system. ISP "A"'s ONIX system then starts a communication with the ONIX system of ISP "B", as shown in step "3", self-advertising the metric capability of that router's link weights. The intention is that the routing tables RIB and FIB on the routers of the other operator simultaneously change without altering the traffic flow. Thus, it is intended that the routing information is modified before the protocol algorithms detect a change and converge into the new topology situation.

Once the new metrics have been exchanged, both ONIX systems inform their respective NET_LEVEL_RM_DEs about the obtained capabilities, represented in step 4. After that, both NET_LEVEL_RM_DEs take the decision to change the routing tables to isolate the affected router and push the commands into the NODE_MAIN_DEs of those involved routers. This triggered action, shown in

step "5", is synchronized between NET_LEVEL_RM_DEs thanks to the ONIX systems.

The involved NODE_MAIN_DEs will push the changes to the corresponding FUNCTION_LEVEL_RM_DEs. These Function-Level-DEs have as associated MEs the routing tables RIB and FIB, thus the intention is to modify the entries maintained by OSPFv3 and BGPv4. This modification will eliminate the traffic flowing through the connections marked with a red cross in the Figure 5.1, allowing just the alternative path represented with dark-green in the same figure (step "6").

At the end, because of the Auto-Collaboration achieved with the Self-Description and Self-Advertisement capabilities, the Self-Adaptation of the network has prevented the fault from appearing. Moreover, although the path between the content provider and the end user has changed, the traffic flow has not been altered. At this moment, the damaged router is isolated, and further administrative decisions can be taken for replacing it.

5.2 Design of the Autonomic Behaviour

In this section the proposed Autonomic Behaviour will be designed. For designing the scenario the Model-Driven Methodology and the Tool-Chain discussed will be applied. Ultimately, it is intended to evaluate the results of the designed system.

Three main steps are needed for the evaluation of the system. The first one is modelling the structure of the Autonomic Behaviour with GME. This structural model will be based on the GANA Meta-Model defined by the EFIPSANS project. In the second step, the resulting model of our system together with the GANA Meta-Model will be transformed to Ecore models. The Ecore models will be used for creating the VBA module, which will be imported into the Visio/M2Code environment. Once the Visio/M2Code environment is established, the behaviour of the system will be modelled with MSCs.

In the third step, the obtained MSCs will be transformed into FSMs. These resulting FSMs are transformed and included in the Ecore model. The infor-

mation of the system's FSMs will be used to create the UPPAAL project file, where the simulation and validation of the system will take place.

Structural Modelling

The structural modelling of the Autonomic Behaviour is done in GME. This application offers the possibility to model the structure of the systems by dragging and dropping graphical blocks. Each block represents a GANA Meta-Model element, and thus any architectural design created with these blocks will conform to the specified meta-model.

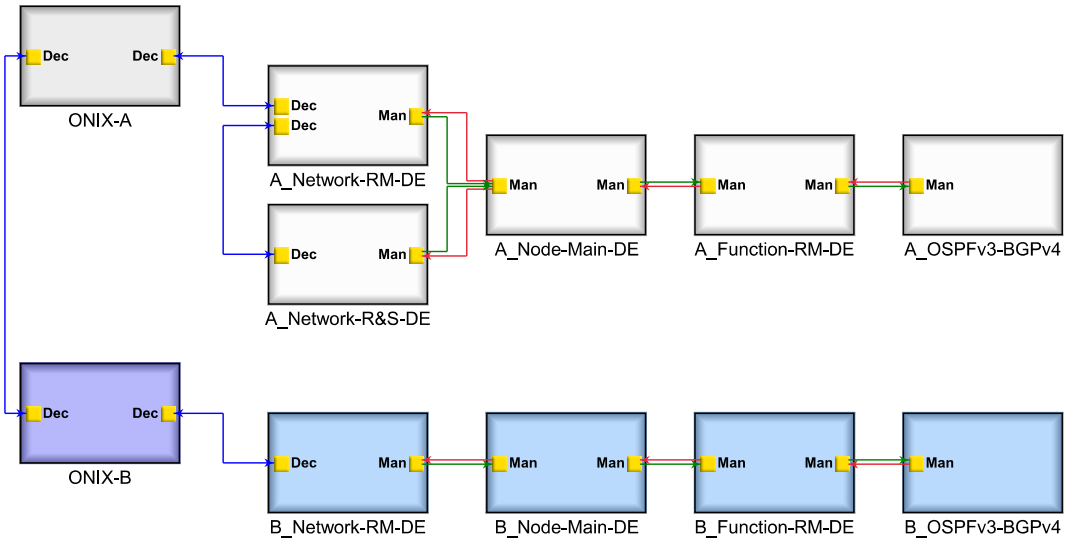

Figure 5.2: Structural Model of the scenario defined in GME

One single Autonomic Behaviour system is created in the GME work space. This system, represented in Figure 5.2, is divided in two different parts, each one representing one of the ISPs involved. These two parts are interconnected through the ONIX systems. In addition, three levels of DEs (Network, Node and Function) and the Protocol level MEs are defined. These MEs represent the entries of the routing tables RIB and FIB maintained by the routing protocols OSPFv3 and BGPv4.

Only the necessary interfaces in the DEs have been included, i.e. the "Manager Interfaces" towards the MEs (represented with green line in the figure), and the "Managed Interfaces" when acting as ME towards the DEs (represented with

red line in the figure). In addition, interfaces for the DE-to-DE, ONIX-to-ONIX and NetworkLevelDE-to-ONIX communications are also included (represented with blue lines).

Model Transformations

Before being able to open the M2Code/Visio environment for modelling the behaviour of our system, the required model transformations need to be processed. This is accomplished by the orchestration of the services inside ModelBus. Therefore, what is needed is to check-in the model created and the GANA Meta-Model file into the ModelBus repository. Both files are used as input parameters of the *GME-Ecore Bridge* transformer, creating as outputs the equivalent models in Ecore technology. These two Ecore models (the GANA Meta-Model and the system model) are used as input files to the *Ecore-to-M2Code* transformation. This transformation creates a VBA module to be imported into the M2Code/Visio project. This module is responsible for ensuring that the MSCs in M2Code conform to the designed model.

Behavioural Modelling

We can now start our behavioural modelling task, which starts by initializing the M2Code application that also launches the Visio environment. We need to create a M2Code project in the M2Code/Visio environment, providing the project's name and the storing folder in the local machine. Using the ModelBus buttons in the Visio toolbar we can launch the generic ModelBus client interface. This graphical interface allows us to browse the ModelBus repository and check-out the VBA module into our local directory. Once this is accomplished, it must be imported into the M2Code project. We are now ready for starting the behavioural modelling of our system.

As described in Section 4.2.2, the module is based on three functions. Initially, the function *CreateControlLoops* needs to be run. This function will create the working sheet where we can start the modelling process.

In Figure 5.3 the modelled behaviour representing the requirements of our system is shown. This figure is divided in two parts: the ISP "A" and the ISP "B" networks. To begin with, all the elements in both networks are in an idle

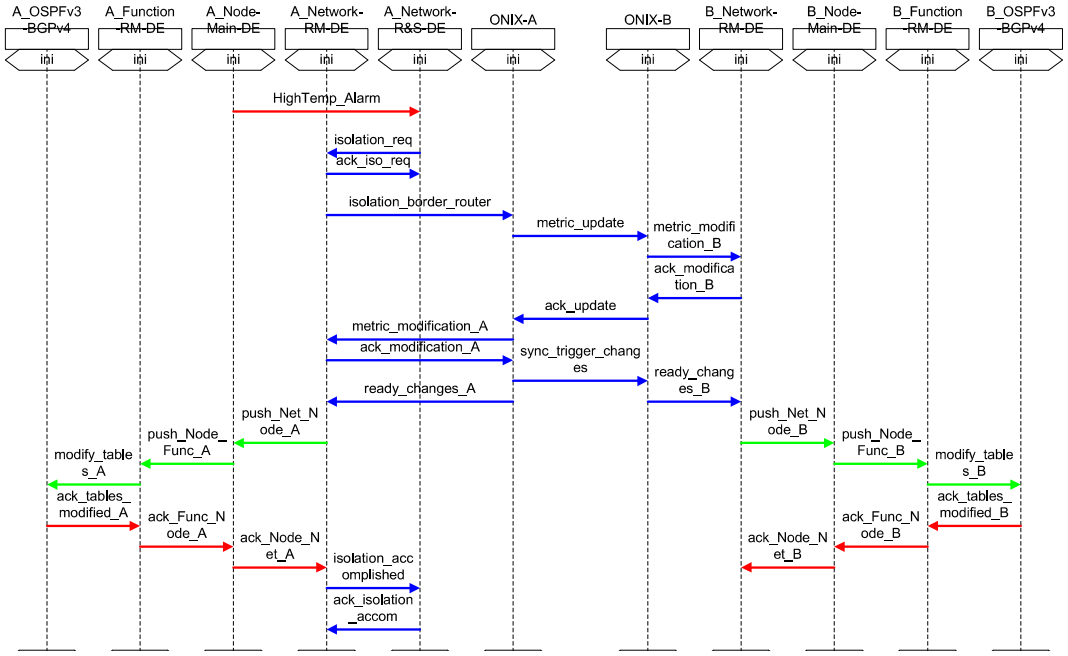

Figure 5.3: MSC model of the scenario defined

state. The message that triggers the functionality of this Autonomic Behaviour is the *HighTemp_Alarm* from the affected router (A_Node-Main-DE) to the Resilience&Survivability Network-Level-DE (Net-R&S-DE). After evaluating the critical status of the router, this DE decides to isolate it, and thus it informs the Routing Management Network-Level-DE (Net-RM-DE) with the *isolation_req* message. The Net-RM-DE checks if the isolation is viable, sending a positive confirmation acknowledgement *ack_iso_req* to the Net-R&S-DE.

Because the Net-RM-DE knows that the affected router is a border router, it decides to contact the other involved network for reaching a collaborative solution. For this purpose, the Net-RM-DE contacts the ONIX system with the request message *isolation_border_router*. The ONIX of ISP "A" sends a self-advertisement message (*metric_update*) to the ONIX at ISP "B". With this message it describes a high metric weight in the links of the router, which implies that it is becoming unreachable. The ONIX "B" informs its own Net-RM-DE about the situation through the message *metric_modification_B*, which is acknowledged back with *ack_modification_B*. In this last message, the Net-RM-DE is informing the ONIX "B" of the change of metrics in its own site, which needs to be sent to ONIX "A". This transfer is done in the message *ack_update*

sent from ONIX "B" to ONIX "A". ONIX "A" forwards this information to the Net-RM-DE through *metric_modification_A* and receives *ack_modification_ A* as answer the message.

At this stage both ONIX systems have self-advertised their respective metric changes, and both Net-RM-DEs know about the modifications needed in the routing tables.

To synchronize the routing-table changes at both sites, the ONIX "A" sends the *sync_trigger_changes* to the ONIX "B", which instructs both Net-RM-DEs to start the processes after receiving the *ready_changes* message.

From this stage, both Net-RM-DEs will follow the same procedure. First of all, they will decide which routers need the changes and what kind of changes they need. After finding the suitable solution, they push the needed changes to the MEs with the *push_Net_Node* command. These MEs are the Node-Main-DEs of the involved routers.

The routers involved in the operation are now aware of the necessary changes and so through the Node-Main-DEs inform the Routing Management Function-Level-DEs (Func-RM-DE) about the received instructions. The MEs of these Func-RM-DEs are the entries in the routing tables maintained by the routing protocols OSPFv3 and BGPv4. This means that the Func-RM-DEs can directly modify, create or delete the entries in the routing tables, a task that is achieved by sending the *modify_tables* messages.

Once the changes in the routing tables are simultaneously completed, the paths crossing the damaged router are eliminated, and only the alternative paths are followed. The confirmation of the changes is sent up to the Net-RM-DEs by crossing the DEs in between. As a final stage, Net-RM-DE sends a confirmation of isolation to the Net-R&S-DE (*isolation_accomplished*), which acknowledges the results with the *ack_isolation_accom* message.

MSC to FSM transformations

The MSC model describes the behaviour we wanted to provide to our system based on the requirements. This MSC model can be transformed to individual FSMs models, i.e. one FSM for each entity (DE, ME or ONIX). This *MSC-to-*

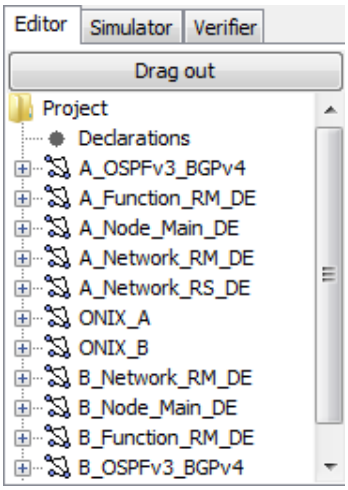

Figure 5.4: UPPAAL Project Elements

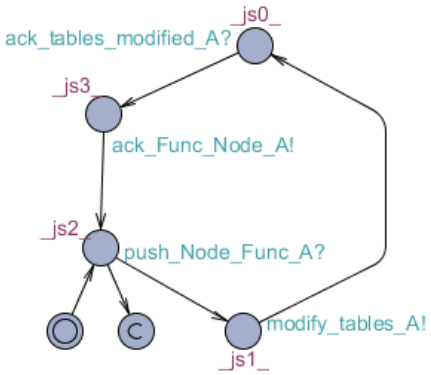

Figure 5.5: Example of FSM in UPPAAL representing Function-RM-DE

FSM transformation takes place in the M2Code environment, which creates a series of files containing the FSM information.

This collection of files is checked-in into the ModelBus repository, where a new transformation is carried out. This new transformation converts the FSM of each entity into an Ecore based model. This model is later included within the whole Ecore model inside each entity's characteristics. The reason behind this transformation is that each application communicates with ModelBus in Ecore based models. If new applications are then connected to ModelBus, they will understand the exchanging process and the base language: Ecore.

Simulation and Validation of the System

The final transformation of the tool-chain is between Ecore and UPPAAL project file. The UPPAAL projects are described with a XML based file, where each entity (DE, ME or ONIX) is described as a FSM template. Thus, it is required that each FSM template contains the same FSM information as the one stored in the Ecore model. Once the project is built with the Ecore model, the simulation, validation and verification of the system can be done.

Figure 5.4 shows the "Editor" tab of the UPPAAL application with our system's elements and Figure 5.5 the FSM of the Function-RM-DE of ISP "A". In this latter figure the messages exchanged by the Function-RM-DE entity can be seen. Each message is marked with a "?", indicating that the entity receives this message; or marked with "!" indicating that the entity sends this message. This way of marking the messages is used by UPPAAL to synchronize the FSMs of different entities. For example, in Figure 5.5 can be observed that this Function-RM-DE will stay in the state "_js2_" until the message *push_Node_Func* is received, changing to state "_js1_". The change from state "_js1_" to state "_js0_" happens after the entity sends the *modify_tables_A* to the ME, which would also trigger a state change in the receiver ME.

By using this synchronization process between FSM/entities, the UPPAAL application allows us to simulate and validate our system. The simulation is performed in the "Simulation" tab, where an automatic process simulates the exchange of all messages. If the design of the system is properly done, the simulation will create a MSC between elements very similar to the one presented in Figure 5.3. One of the reasons allowing us to assume that our system is validated is if it behaves as expected, i.e. if during the simulation process no communication deadlocks are found.

Simulation and Validation Results

When applying the UPPAAL mechanisms to our designed system, the simulation accomplished the automatic process without any deadlock. According to this validation result, we can conclude that our system and its behaviour were successfully designed, and that no inconsistencies were introduced during the modelling process.

On the other hand, no verification of the system was done using the UPPAAL application. The verification of the system would offer one deeper step into the analysis of the system's technology; a step that the current model information does not allow to be performed. Through this verification process it could be proven if, for example, timers, loops, etc. behave correctly in our system. As it is observed, those kinds of variables are not modelled in any of the tools of the tool-chain, and thus no verification of them is needed. Further discussion on this topic is covered in Chapter 6.

If we consider issues related to the stability of the Autonomic Behaviour, we can say that several approaches were taken for addressing some stability threats through the whole modelling process. The three structural solutions during the design-period for avoiding instability in the system were respected. In other words, the hierarchical control loops within each DE have separated the working abstract levels of the network. Additionally, the concept of "ownership" has been followed as just one DE has taken actions into a single ME at any given time. Furthermore, the separation of operational regions has been achieved as the Routing Management and the Resilience and Survivability management were separated in different DEs.

In addition to the design-period solutions, a runtime solution was added. This solution was based on the synchronization between both networks, achieved by the collaborative communication between both ONIX systems. The accomplished synchronized execution of the tasks in both networks allows the stability concepts to be maintained in the running system.

5.3 Summary

In this Chapter we have designed an Autonomic Behaviour to be run in a networking scenario. The networking scenario described is based on the usage of Self-Adaptation, Seft-Advertisement, Self-Description and Auto-Collaboration capabilities for avoiding a failure to appear in the network.

While monitoring the network, a high risk signal is raised from one border gateway router, which triggers the Autonomic Behaviour to start. The aim of this Autonomic Behaviour is to divert the traffic crossing the affect router to alternative paths, accomplishing the isolation of the router without altering the traffic. The Autonomic Behaviour will involve in the process routers from two different ISPs.

The design of our Autonomic Behaviour started by modelling its structure in GME. Later, we modelled the behaviour of the DEs involved with a MSC. This MSC model was transformed into FSMs, which were converted into an UPPAAL project file. In UPPAAL we successfully simulated and validated our system. Therefore, our system was designed and modelled without inconsistencies.

Chapter 6

Conclusion

The benefits obtained by applying Model-Driven techniques for the development of complex systems are well known within the software engineering world, above all by engineers following the modelling engineering approach. For that reason, it would be beneficial if networking engineers become aware of these kinds of solutions, being able to apply such techniques in the development of networking architectures.

The Model-Driven Methodology proposed appears to be a viable solution for approaching the design of Autonomic Behaviours. During the first phase of the methodology's development, a deep analysis of network and system requirements was performed. This analysis intended to propose solutions for addressing structural and behavioural complexity and stability issues. Hierarchical solutions allow both the complexity and stability at different abstract levels to be addressed, which makes it easier to conceptually separate, design and develop those hierarchical levels. Furthermore, we believe that the high level of importance provided to tackle stability issues within the EFIPSANS project has enhanced the final value of the project's outcome. In such complex systems, where the entities are distributed and working on parallel time-scales, instability is one of the biggest problems that may appear. For this reason, addressing this problem from the beginning of the project would allow GANA to obtain a strong architectural base for deploying autonomic networks.

Within the evolution of this thesis, we brought up several matters to be discussed. While using the tools of the tool-chain we evaluated the GANA Meta-

Model, where it was found that some potential enhancements could be included. For instance, if we check the part of the GANA Meta-Model describing the FSMs, shown in Figure 3.10, we can see that the transitions have a parameter *transitionID* used for identifying each transition. However, in addition to the *transitionID*, a transition message that would record the message exchanged between states in the FSMs is needed. This parameter is a requirement for allowing the transformation between Ecore and UPPAAL application file.

The developed tool-chain is a very interesting solution for the methodology proposed, as it is a great advantage to be able to use services offered by different tools as a unique tool. It is a well-known time-consuming task having to manually transform models to be used by different tools, which could also introduce some errors in the models by mistake. Thus, providing a solution where the orchestration of services is carried out by a method of integration framework may be considered a great achievement. This benefit could be realized by using the ModelBus application.

Alternatively, some improvements that could be introduced into the tool-chain were identified. One of these improvements is based on the comments added in the case study Chapter 5, where the UPPAAL service "verification" was not used due to the non-existence of variable parameters that would drive the FSM behaviour.

The M2Code application allows the creation of FSMs based on MSCs; however, it is not possible to address a deep system complexity with this application. The reason is that the limitations of this application do not allow for modelling of the DE's internal processes; for example, it is not possible to send messages from one DE to itself.

The output of this application is based on representing the FSM as driven by the exchange of messages, i.e. the FSMs represent the communication process of the DEs with other entities. However, within the proposed tools there is no suitable solution for defining the actions to be done on each FSM state/transition. Therefore, because these kinds of definitions need to be done outside of the tool-chain, we are certain that this situation is not the most suitable. It can be said, for summarizing this situation that addressing a deep system complexity by using M2Code does not seem to be viable.

For that reason, the tool-chain could be enhanced with a tool that would allow us to model the systems with a higher complexity approach. For example, a Specification and Description Language (SDL) based tool would improve the tool-chain in such a way that systems can be modelled with more accurate techniques. By creating SDL diagrams the systems can be modelled using more complex logic, such as introducing timers, conditions, loops, etc.

Furthermore, we think that a tool that could create FSM models out of SDL diagrams would be the best improvement in this matter. Later, these kinds of FSMs containing variable parameters could be transformed into UPPAAL systems, and thus more accurate simulation, validation and verification of our systems could be performed.

The tool adapters designed and developed for achieving some model transformations provided successful results according to the requirements. The development of the transformations was accomplished by using EMF libraries managing the Ecore models. This way of creating, updating and modifying models has a lot of benefits when the management of models representing complex systems can be used, for example, for testing purposes. On the other hand, we believe that other functionalities could have been developed and included in the tool-chain transformations that would have allowed the process of the models to be more automatic. For example, a script could automate the situation where the user has to manually import the VBA module into the local drive for being manually included in the M2Code application later. This script could automatically achieve this task without needing any human intervention.

In addition, a small limitation is seen in the flow of the models. In the Visio framework we are able to model the behaviour of the DEs with MSCs; however, the transformation from MSC to FSM happens internally in the Visio/M2Code environment. This situation restricts the ModelBus's awareness of the MSC model. Therefore, an enhancement to the tool-chain could be introducing a tool adapter that would transform the MSC model to Ecore based language in case another tool could use this information as input for further modelling of the MSC diagrams.

As a conclusion about the applicability of a Model-Driven Methodology and a Tool-Chain for the design of Autonomic Behaviour, the proposed solution turned out to be a suitable approach for the level of complexity addressed. On the other hand, we cannot assume any potential results that could be obtained

when addressing more complex systems, such as those where hundreds of nodes might be involved in the Autonomic Behaviour.

This methodology, together with the tool-chain, allows us to dynamically design Autonomic Behaviours without having to take care of some conflicts that may appear during the design-period. The tools used in the tool-chain provide the users with the intrinsic capacity of following the needed requirements for designing this kind of complex systems without needing to know about them. An example of these followed requirements is the inclusion of solutions that address stability threats, such as the hierarchical structure of the DEs coming from the GANA Meta-Model. However, we consider that there is room for some enhancements in the methodology, such as the addition of an SDL application to the tool-chain for the purpose of modelling the internal processes of the DEs.

Future work

The EFIPSANS project has studied and provided a great deal of documentation in regards to creating autonomic networks based on the GANA architecture. Although several Autonomic Behaviours were demonstrated in a test plan, it is clear that a future project would be to deploy a whole GANA network from scratch.

We have demonstrated that by applying the proposed Model-Driven Methodology Autonomic Behaviours can be designed, simulated and validated. Thus, for proving the benefits of using the methodology and the tool-chain a real Autonomic Behaviour, designed using this approach, could be fully developed, deployed and integrated into a network. Accomplishing the task of developing a running Autonomic Behaviour, which was designed using the methodology, would enhance the total value of the methodology proposed.

In addition, and as discussed in the previous section, an enhancement of the tool-chain could be studied. Some additional tools providing alternative solutions could be integrated into the tool-chain allowing a more complex system modelling. Moreover, the tool-chain could also be enhanced with a tool offering code-generation. After designing the Autonomic Behaviour, this tool could be used to create code representing the skeleton and some of the logic of the entities involved in the system.

Bibliography

[1] International Telecommunication Union Telecommunications Sector (ITU-T). Principles for a Telecommunications Management Network. Recommendation M.3010, 1996.

[2] J. Richard Burke. *Network Management: Concepts and Practice, A Hands-On Approach*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2003.

[3] 3rd Generation Partnership Project (3GPP). Telecommunication Management: Principles and High Level Requirements. Specification TS 32.101 V9.1.0, 3GPP, April 2010.

[4] European Union Information Society Technologies Framework Programme 6 (EU IST FP6). Autonomic Network Architecture (ANA). [Online]. Available: http://www.ana-project.org/web/, Accessed: April 2011.

[5] EFIPSANS. Exposing the Features in IP version six Protocols that can be exploited/extended for the purposes of designing/building Autonomic Networks and Services. [Online]. Available: http://www.efipsans.org/index.php, Accessed: April 2011.

[6] Telefonica España. Proyecto europeo: EFIPSANS - Operational Challenges. Internal EFIPSANS documentation, January 2008.

[7] Ranganai Chaparadza (Ed.). Second Draft of Autonomic Behaviours specifications (ABs) for Selected Diverse Networking Environments: Core Draft Document. Technical Report INFSO-ICT-215549/WP1/D1.3v1, EFIPSANS, December 2008.

[8] Albert Greenberg, Gisli Hjalmtysson, David A. Maltz, Andy Myers, Jennifer Rexford, Geoffrey Xie, Hong Yan, and Jibin Zhan & Hui Zhang. A

Clean Slate 4D Approach to Network Control and Management. *ACM SIGCOMM Computer Communication Review*, October 2005.

[9] Hitesh Ballani and Paul Francis. CONMan: Taking the Complexity out of Network Management. In *Proceedings of the 2006 SIGCOMM workshop on Internet network management*, INM '06, pages 41–46, New York, NY, USA, 2006. ACM.

[10] John Strassner, Nazim Agoulmine, and Elyes Lehtihet. *FOCALE - A Novel Autonomic Computing Architecture*. 2006.

[11] Ranganai Chaparadza. EFIPSANS - Spirit and Vision. [Online]. Available: http://www.efipsans.org/images/stories/EFIPSANSAccessed: April 2011.

[12] Ranganai Chaparadza. Self-Management Activities in the EFIPSANS Project. [Online]. Available: http://www.efipsans.org/images/stories/EFIPSANSop.pdf, Accessed: April 2011.

[13] EFIPSANS Project. Periodic Management Reports. Internal Document, June 2009.

[14] Ranganai Chaparadza (Ed.). Final Draft of Autonomic Behaviours specifications (ABs) for Selected Diverse Networking Environments (Core Document). Technical Report EFIPSANS/CO/WP1/D1.7/Part1/v1.0, EFIPSANS, 2010.

[15] Ranganai Chaparadza and Arun Prakash. Requirements for Model-Driven Design Methodology and Ontologies for GANA Based Networks. Technical Report EFIPSANS/CO1/WP1/D1.6/SD/v1.0, EFIPSANS, December 2009.

[16] ISG AFI. Generic Autonomic Network Architecture: An Architectural Reference Model for Autonomic Networking, Cognitive Networking and Self-Management. Technical Report Draft ET GS AFI 002 V0.0.11 (2011-10), ETSI, 2011.

[17] Peter Benko, Vassilios Kaldanis, and Domonkos Asztalos. EFIPSANS Integration Framework. Technical Report EFIPSANS/D5.1v7, EFIPSANS, 2010.

[18] Roger Pressman. *Software Engineering: A Practitioner's Approach*. McGraw-Hill, Inc., New York, NY, USA, 7 edition, 2010.

[19] Arun Prakash, Ranganai Chaparadza, and Zoltan Theisz. Requirements of a Model-Driven Methodology and Tool-Chain for the Design and Verification of Hierarchical Controllers of an Autonomic Network. In *Proceedings of the 2010 Third International Conference on Communication Theory, Reliability, and Quality of Service*, CTRQ '10, pages 208–213, Washington, DC, USA, 2010. IEEE Computer Society.

[20] Object Management Group (OMG). OMG's Metaobject Facility. [Online]. Available: http://www.omg.org/mof/, Accessed: April 2011.

[21] David Steinberg, Frank Budinsky, Marcelo Paternostro, and Ed Merks. *EMF: Eclipse Modeling Framework 2.0*. Addison-Wesley Professional, 2nd edition, 2009.

[22] Eclipse Ganymede. Eclipse documentation - the Eclipse Modeling Framework (EMF) overview. [Online]. Available: http://help.eclipse.org/ganymede/index.jsp?topic=/org.eclipse.emf.doc /references/overview/EMF.html, Accessed: April 2011.

[23] Institute for Software Integrated Systems. Vanderbilt University. GME: Generic Modeling Environment. Copyright 2011. [Online]. Available: http://www.isis.vanderbilt.edu/Projects/gme, Accessed: May 2011.

[24] Arun Prakash and Ranganai Chaparadza. Model-Driven Methodology and Associated Tool-Chain for Design for Stability in GANA. Technical Report INFSO-ICT-215549/EFIPSANS/CO/WP1/D1.8b/v1.0, EFIPSANS, 2011.

[25] Hakan Coskun and Reinhard Ruppelt. Stability and Scalability within EFIPSANS. Technical Report EFIPSANS/CO1/WP1/D1.5b/v1.0, EFIPSANS, 2009.

[26] Gerard J. Holzmann. *Design and Validation of Computer Protocols*. Prentice Hall, November 1990.

[27] Fraunhofer Institude for Open Communication Systems FOKUS. Model Bus. [Online]. Available: http://www.modelbus.org/modelbus/, Accessed: May 2011.

[28] ModelBus Team. *ModelBus Users Guide*. Fraunhofer FOKUS, February 2011.

[29] Microsoft Corporation. Visio 2010. Copyright 2011. [Online]. Available: http://office.microsoft.com/en-us/visio/, Accessed: May 2011.

[30] Ingolf Krüger. *Distributed System Design with Message Sequence Charts*. PhD thesis, Technische Universität München, 2000.

[31] Praveen. N. Moorthy. Building a Tool for Synthesis of Correct Design from Interaction Specifications. Master's thesis, University of California, San Diego, 2006.

[32] Up4All International AB. Up4All UPPAAL v4 the Simulations Platform. Copyright 2009. [Online]. Available: http://www.uppaal.com/, Accessed: April 2011.

[33] Gerd Behrmann, Alexandre David, and Kim G. Larsen. A tutorial on UPPAAL. Department of Computer Science, Aalborg University, Denmark. november 2004. [Online]. Available: http://www.it.uu.se/research/group/darts/papers/texts/new-tutorial.pdf, Accessed: May 2011.

[34] The MathWorks. Matlab and Simulink for Technical Computing. [Online]. Available: http://www.mathworks.com/, Accessed: May 2011.