Abstract: Autol project envisages Future Internet of Networks and Service as an optimised network and service solution, which guarantees built-in orchestrated security, reliability, robustness, mobility, context, access, service support and self-management of the communication resources and services. A new autonomic management architectural model for Future Internet is presented in this document.
**Document Properties**

<table>
<thead>
<tr>
<th>Document Number:</th>
<th>FP7-ICT-2007-1-216404/D3.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Document Title:</td>
<td>Information Model</td>
</tr>
<tr>
<td>Document responsible:</td>
<td>Zohra Boudjemil (WIT)</td>
</tr>
<tr>
<td>Author(s) / Editor(s):</td>
<td>Zohra Boudjemil(WIT), Steven Davy(WIT), David Muldowney(WIT), Claire Fahy(WIT)</td>
</tr>
<tr>
<td>Reviewed by:</td>
<td>Alex Galis(UCL)</td>
</tr>
<tr>
<td>Dissemination Level¹:</td>
<td>Public Information</td>
</tr>
<tr>
<td>Status of the Document:</td>
<td>Final</td>
</tr>
<tr>
<td>Version:</td>
<td>2.0</td>
</tr>
</tbody>
</table>

This document has been produced in the context of the Autonomic Internet (AutoI) Project. The AutoI Project is part of the European Community’s Seven Framework Program for research and is as such funded by the European Commission. All information in this document is provided “as is” and no guarantee or warranty is given that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability. For the avoidance of all doubts, the European Commission has no liability in respect of this document, which is merely representing the authors view.

¹Dissemination level as defined in the EU Contract:
PU = Public
PP = Distribution limited to 6th FP participant
## Revision History

<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Issued by</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>06/10/08</td>
<td>Zohra Boudjemil</td>
<td>ToC</td>
</tr>
<tr>
<td>0.2</td>
<td>20/10/08</td>
<td>Claire Fahy</td>
<td>Service specifications</td>
</tr>
<tr>
<td>0.3</td>
<td>28/10/08</td>
<td>Steven Davy</td>
<td>Policy specifications</td>
</tr>
<tr>
<td>0.4</td>
<td>07/11/08</td>
<td>Zohra Boudjemil</td>
<td>CIM description, Resource specifications, Context specifications, Root specifications</td>
</tr>
<tr>
<td>0.5</td>
<td>16/11/08</td>
<td>Claire Fahy</td>
<td>Review/Updates to Service Chapter</td>
</tr>
<tr>
<td>0.6</td>
<td>19/11/08</td>
<td>Zohra Boudjemil</td>
<td>AIM Domains, Root Entities, Resources (Physical and Logical)</td>
</tr>
<tr>
<td>0.7</td>
<td>22/11/08</td>
<td>Claire Fahy</td>
<td>Updates to Service Lifecycle model/Rearrangement of Chapter.</td>
</tr>
<tr>
<td>0.8</td>
<td>24/11/08</td>
<td>Zohra Boudjemil</td>
<td>Merged copy from v0.6 and v0.7 (Claire and Steven version), plus David’s comments</td>
</tr>
<tr>
<td>0.9</td>
<td>28/11/08</td>
<td>Zohra Boudjemil</td>
<td>virtualResource, reorganised Management section</td>
</tr>
<tr>
<td>0.10</td>
<td>01/12/08</td>
<td>Zohra Boudjemil</td>
<td>Topology, Graph, StateMachine, Service, Service Resource relationships and Service Instantiation from Claire’s version</td>
</tr>
<tr>
<td>1.0</td>
<td>01/12/08</td>
<td>Zohra Boudjemil</td>
<td>Virtualisation Instantiation</td>
</tr>
<tr>
<td>1.1</td>
<td>02/12/08</td>
<td>Zohra Boudjemil</td>
<td>Policy Goals and SID summery from Steven, Structure of the Document, updates from the first review</td>
</tr>
<tr>
<td>1.2</td>
<td>03/12/08</td>
<td>Zohra Boudjemil</td>
<td>Executive summary by David, updates to the State of the art introduction and add new section for the Major AutoI modelling requirements</td>
</tr>
<tr>
<td>1.3</td>
<td>08/12/08</td>
<td>Zohra boudjemil</td>
<td>Information in relation to the Policy continuum, conclusion, updates relating to the internal reviews.</td>
</tr>
<tr>
<td>1.4</td>
<td>17/12/08</td>
<td>Zohra Boudjemil</td>
<td>References, updates relating to the internal reviews</td>
</tr>
<tr>
<td>1.5</td>
<td>22/12/08</td>
<td>Alex Galis</td>
<td>2nd review cycle</td>
</tr>
<tr>
<td>1.6</td>
<td>05/01/09</td>
<td>Zohra Boudjemil</td>
<td>Updates following the 2nd review cycle</td>
</tr>
<tr>
<td>1.7</td>
<td>09/01/09</td>
<td>Zohra Boudjemil</td>
<td>Formatting (look&amp;feel) changes based on D6.1 template</td>
</tr>
<tr>
<td>2.0</td>
<td>12/01/09</td>
<td>Zohra Boudjemil</td>
<td>Final version</td>
</tr>
</tbody>
</table>
Table of Contents

DOCUMENT PROPERTIES ........................................................................................................... 3

REVISION HISTORY ..................................................................................................................... 4

TABLE OF CONTENTS .................................................................................................................. 5

TABLE OF FIGURES ..................................................................................................................... 7

EXECUTIVE SUMMARY .............................................................................................................. 8

1 INTRODUCTION ....................................................................................................................... 10
  1.1 DOCUMENT SCOPE ............................................................................................................ 10
  1.2 STRUCTURE OF THE DOCUMENT .................................................................................... 10
  1.3 OBJECTIVES ..................................................................................................................... 10

2 STATE OF THE ART: INFORMATION MODELS FOR COMMUNICATION NETWORKS ............ 11
  2.1 MAJOR AUTOI MODELING REQUIREMENTS .................................................................. 11
  2.2 THE COMMON INFORMATION MODEL, CIM ................................................................ 12
  2.3 THE SHARED INFORMATION AND DATA MODEL, SID .................................................. 14
  2.4 DIRECTORY ENABLED NETWORK-NEW GENERATION, DEN-NG ................................. 15

3 AUTOI INFORMATION MODEL, AIM .................................................................................... 17
  3.1 MODEL SPECIFICATIONS .................................................................................................. 17
    3.1.1 AIM Design Process .................................................................................................. 18
    3.1.2 Root Entities ............................................................................................................. 18
    3.1.3 Resources ................................................................................................................ 19
      3.1.3.1 NonVirtualResource ......................................................................................... 20
      3.1.3.2 VirtualResource ............................................................................................... 22
    3.1.4 Services .................................................................................................................... 23
      3.1.4.1 Basic Service Concepts .................................................................................... 24
      3.1.4.2 Service Lifecycle Model Concept ..................................................................... 25
      3.1.4.3 Service and Resource Relationship ................................................................. 26
    3.1.5 Management Concepts ............................................................................................... 27
      3.1.5.1 Policy Domain .................................................................................................. 27
        3.1.5.1.1 The Basics ................................................................................................ 27
        3.1.5.1.2 PolicyRuleStructure ................................................................................. 27
        3.1.5.1.3 PolicyRuleComponentStructure ............................................................... 28
        3.1.5.1.4 Policy Categories ...................................................................................... 29
        3.1.5.1.5 Policy Applications ................................................................................... 32
        3.1.5.1.6 AutoI Policy Continuum ........................................................................... 33
    3.1.5.2 Context .................................................................................................................... 35
      3.1.5.2.1 Context aware policy model ....................................................................... 37
    3.1.5.3 MetaData ................................................................................................................ 38
      3.1.5.3.1 Roles ............................................................................................................ 38
      3.1.5.3.2 Capabilities .................................................................................................. 38
      3.1.5.3.3 Capacities .................................................................................................... 39
    3.1.5.4 Topology ................................................................................................................ 39
    3.1.5.5 State machine model for management ................................................................. 41
  3.2 INSTANTIATION EXAMPLES .............................................................................................. 41
    3.2.1 Resources Instantiation ............................................................................................. 42
    3.2.2 Services Instantiation ................................................................................................ 42

4 USAGE OF THE AUTOI INFORMATION MODEL .................................................................... 44
  4.1 MODEL DRIVEN PROCESS ............................................................................................... 44
    4.1.1 Process step: IDENTIFICATION ............................................................................ 44
    4.1.2 Process step: TRANSFORMATION ....................................................................... 45
    4.1.3 Process step: GENERATION .................................................................................. 45
  4.2 INFORMATION PART OF THE AUTOI INTERFACES ......................................................... 46
CONCLUSION ................................................................................................................................. 48
REFERENCES ......................................................................................................................................... 50
ANNEX ................................................................................................................................................. 52
  7.1 CIM SPECIFICATION RESOURCES ...................................................................................... 52
  7.2 MODELLING REQUIREMENTS ................................................................................................. 53
  7.3 AIM SPECIFICATION RESOURCES ....................................................................................... 55
    7.3.1 AIM Domains dependencies ............................................................................................... 55
    7.3.2 ManagedEntity Relationships to ManagementDomain and ManagementApplication .......... 56
    7.3.3 ManagedEntity Subclasses .................................................................................................. 57
    7.3.4 Identity and Identification ..................................................................................................... 58
    7.3.5 PhysicalResources ................................................................................................................ 59
    7.3.6 LogicalResources .................................................................................................................. 59
    7.3.7 Graph entities ......................................................................................................................... 60
# Table of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CIM Virtualization, Resource Pools and Allocation</td>
</tr>
<tr>
<td>2</td>
<td>SID Overview</td>
</tr>
<tr>
<td>3</td>
<td>Root Entities Overview</td>
</tr>
<tr>
<td>4</td>
<td>Resource Concepts Overview</td>
</tr>
<tr>
<td>5</td>
<td>VirtualResource Overview</td>
</tr>
<tr>
<td>6</td>
<td>VirtualDeviceInterface &amp; VirtualRouter</td>
</tr>
<tr>
<td>7</td>
<td>Basic Service Concepts</td>
</tr>
<tr>
<td>8</td>
<td>AutoI Service States</td>
</tr>
<tr>
<td>9</td>
<td>Service Resource Relationships</td>
</tr>
<tr>
<td>10</td>
<td>PolicyRuleStructure</td>
</tr>
<tr>
<td>11</td>
<td>PolicyRuleComponentStructure</td>
</tr>
<tr>
<td>12</td>
<td>ManagementPolicy</td>
</tr>
<tr>
<td>13</td>
<td>PolicyCategory</td>
</tr>
<tr>
<td>14</td>
<td>AIM PolicyCategories</td>
</tr>
<tr>
<td>15</td>
<td>ManagementApplication</td>
</tr>
<tr>
<td>16</td>
<td>ManagementApplicationComponent</td>
</tr>
<tr>
<td>17</td>
<td>Context and ContextData</td>
</tr>
<tr>
<td>18</td>
<td>Relationships between Context and Policy</td>
</tr>
<tr>
<td>19</td>
<td>Topology Overview</td>
</tr>
<tr>
<td>20</td>
<td>Topology defined by Policy</td>
</tr>
<tr>
<td>21</td>
<td>StateMachine Concepts Overview</td>
</tr>
<tr>
<td>22</td>
<td>Resources Instantiation example</td>
</tr>
<tr>
<td>23</td>
<td>Service Example Instantiation</td>
</tr>
<tr>
<td>24</td>
<td>Model Usages</td>
</tr>
<tr>
<td>25</td>
<td>CIM Core Specifications (Overview)</td>
</tr>
<tr>
<td>26</td>
<td>CIM Core Network Specification (Overview)</td>
</tr>
<tr>
<td>27</td>
<td>AIM packages and their dependencies Root Domain Usage</td>
</tr>
<tr>
<td>28</td>
<td>Management domain constituencies</td>
</tr>
<tr>
<td>29</td>
<td>Management Domain usage</td>
</tr>
<tr>
<td>30</td>
<td>ManagedEntity Relationships to ManagementDomain and ManagementApplication</td>
</tr>
<tr>
<td>31</td>
<td>ManagedEntity Subclasses</td>
</tr>
<tr>
<td>32</td>
<td>Identity and Identification Overview</td>
</tr>
<tr>
<td>33</td>
<td>PhysicalResources Overview</td>
</tr>
<tr>
<td>34</td>
<td>LogicalResources Overview</td>
</tr>
<tr>
<td>35</td>
<td>Graph Entities Overview</td>
</tr>
</tbody>
</table>
Executive Summary

The AUTOI project is developing a solution for the manageability of the Future Internet under the concept of five planes (OSKVM) - Virtualisation Plane (VP), Management Plane (MP), Knowledge Plane (KP), Service Enablers Plane (SP) and Orchestration Plane (OP). The overall architectural model of the Autol project was depicted in D6.1 of August 2008 and its update D4.1 of Dec 2008.

This document is directly linked to address the Objectives 3, 4 and 5 of the project work plan, namely the dynamic information base to support the service-oriented autonomic orchestration of resources, the definition of an appropriate policy continuum and the Self-organisation of the virtual resources into overlay solutions (across heterogeneous environments).

This document describes the Information Modelling developed for the Autol project. A key objective of the project is the creation of a virtual communication resource overlay with autonomic characteristics to adapt the services and resources offered to meet changing user needs, business goals, and environmental conditions. Designing a service-aware network, a key research challenge, is realising the capability of self-knowledge which empowers the network to reconfigure itself autonomously in the face of change to adapt its services according to business goals and policies.

Each overlay should be capable of spanning heterogeneous networks and supporting service mobility, security, quality of service and reliability. Managing a diverse network by traditional mechanisms is at best complex and consequently requires costly administration to meet the requirements of a dynamic and diverse service environment. This complexity could be reduced if the network was capable of some autonomic behaviour; however, in order to provide new services, the network will need to be aware of the requirements of the services and how they impact currently provisioned networks – quite a challenge given the heterogeneity of the network equipment.

Autol is creating an overlay of self-organising virtual network resources that self-organises according to service requirements or changes in service context, and the deployed services self-configure in response to changes in the overlay. Orchestration mechanisms mediate between network operations, for example mobility, security and QoS, enabling conflict resolution and harmonisation. This overlay is subsequently realised by the successful co-operation of the following activities: autonomic control principles, resource virtualisation, enhanced control algorithms, information modelling, policy-based management and service programmability.

As a step towards building and maintaining this self-knowledge overlay, the Autol Information Model (AIM) has been defined so that, in conjunction with other knowledge representations (such as ontologies), it provides a common language to represent the concepts, characteristics and behaviour required for the self-management of the overlay. It is aimed at Information Networking of Future Internet. Specifically, the lexicon of the Autol common language enables existing and future network management data to be mapped to the Autol common language.

The Information Model defined in this document serves to capture all necessary concepts concerned with service-oriented virtual resource orchestration and their relationships. An information model can be defined as a “representation of the characteristics and behaviour of a component or system independent of vendor, platform, language, and repository”. The use of a single information model will enable multiple platform- and language-specific data models to be built that each have a common understanding of shared data. The Autol Information Model uses a set of abstractions and software patterns that enable services to
express their needs to the management overlay, which translates those needs into a form that the network can understand.

A subset of the DEN-ng information model [17] is used as the core of the AIM model. Then, new extensions are added to support new concepts such as virtualisation. In addition, some appropriate refinements to the model are done in conjunctions with the AutoI requirements and constraints. Thus, the core of the AIM model is based on a DEN-ng subset.

The AIM model plus a set of domain-specific ontologies can then be used as a common language to advance interoperability and understanding across the disparate components of the AutoI architecture. In the same way, the common language enables network resources to be defined in such a way that the services can work with and use them.

This common language becomes the base for a set of Domain Specific Languages (DSL) which addresses the specific tasks of the framework while still enabling interoperability. Thus, the Information Model described in this document forms the foundation for work in the second half of the project where the ontologies and DSLs specific to AutoI will be specified in detail.

The specifications of the AutoI Information Model (AIM) have been defined in parallel to the definition of the AutoI architecture and the design of the AutoI five planes (OSKMW). Therefore, the refinement of the model is not finished as the design of the planes is still an ongoing task. The use of the AIM in the AutoI project is planned for year 2 of the project.
1 Introduction

1.1 Document Scope
This document should describe the Autol Information Model (AIM) which is the definition that is used to model all of the different entities (including their properties and relationships) of a problem domain or specifically it should define managed entities that are visible across all the planes of the Autol architecture [1], i.e. Orchestration, Knowledge, Management and Virtualisation Planes, and realising the requirement specified in [1].

The state of the art in this research area should be summarised along with the rationale on why DEN-ng was the Information Model of choice for the starting point for the Autol Information Model, describing how DEN-ng was applied to the Autol problem domain, where and why the DEN-ng was extended to support Autol.

The Autol Information Model, as an artefact, is defined using Rational Software Architect, so this document is not intended to be an exhaustive specification of the model. Instead, this document introduces the Information Model putting it into context of Autol by describing the main modelling components required by each of the architectural planes.

This document should describe how the Information Model should be used by the software architectural components in each of the planes in addition to describing the model driven process that will be used to generate the Domain Specific Languages relevant for Autol.

1.2 Structure of the Document
This deliverable is organised as follows:

- Chapter 2 - State of the Art: this chapter provides an overview of the existing Information Models for Communication Networks, and discussions of their usability in the context of Autol.
- Chapter 3 - Autol Information Model: this chapter start by the core specifications of the AIM model. The second part provides instantiation examples of the AIM. The last part analyses the information captured by the AIM and its relation to the policy continuum layers.
- Chapter 4 - Usage of the Autol Information Model: this chapter provides a description on the usage of the AIM model in the Autol project.
- Chapter 5 - Conclusion: this is the summery of the document.

1.3 Objectives
The primary objective of this document is to define a means for establishing a knowledge base and behavioural definition for the purposes of service-oriented resource management by defining a single information model.

This model is required for the derivation of data models specific to a technology or vendor that enables data coherency, understanding and interoperability across heterogeneous environments. This information model will be extended to accommodate the orchestration of resources based on service requirements and to support the concept of policy continuum.
2 State of the Art: Information Models for Communication Networks

An information model is “a representation of concepts, relationships, constraints, rules, and operations to specify data semantics for a chosen domain of discourse. The advantage of using an information model is that it can provide sharable, stable, and organized structure of information requirements for the domain context.” [2]. The information model is an abstract representation of entities which can be real objects such as devices in a network or logical such as the entities used in a billing system. Typically, the information model provides formalism to the description of a specific domain without constraining how that description is mapped to an actual implementation. Thus, different mappings can be derived from the same information model. Such mappings are called data models.

Three major information models are widely used today to capture semantics and behaviour of communication networks: the Common Information Model (CIM), the Shared Information and Data Model (SID), the Directory Enabled Network-new generation (DEN-ng).

For the definition of the information model for the Autol project, there are a set of modelling requirements (cf. section 7.2) that the model has to address. The three existing information models are investigated following these requirements to understand which one is more suitable for the Autol domain and can be used as a core to build the Autol information model.

The first section describes the main modelling requirements which are a summary of the set of modelling requirements defined in D6.1 and also available in section 7.2. The following sections are dedicated to each major existing information model presentation and analysis of their usability for Autol.

2.1 Major Autol Modeling Requirements

The modelling requirement analysis was done at the first steps of the project in relation to the definition of the Autol architecture and the functionalities of each plan. This analysis provided a set of explicit modelling requirements described in D6.1 [1]. The following list of requirements is a short list of the major ones that the available information models have to address to be suitable for the Autol domain:

- The information model has to provide conceptual abstractions of these main manageable entities: resource, service.
- The information model has to support the representation of the service and resource characteristics, their relationships and their dependencies that are relevant to the management functions.
- The information model has to be easily extensible to support new artefacts describing new concepts related to virtualisation
- The information model has to provide concepts to support the representation of behavior of a managed entity such as state machines
- The information model has to provide abstractions supporting the representation of contextual data related to the managed entities.

The modelling requirements are also available in section 7.2.
The information model has to support policy concepts that enable the self-* capabilities of the Autol components and that provide means to the information translations between the levels of the policy continuum.

2.2 The Common Information Model, CIM

The Common Information Model (CIM) is part of the standards defined by the Distributed Management Task Force (DMTF). The DMTF is "a non-for-profit association of industry members dedicated to promoting enterprise and systems management and interoperability." [3]. The DMTF, while being an industrial consortium, allows "members and non-members to reproduce DMTF specifications and documents for uses consistent with this purpose management and interoperability, provided that correct attribution is given. As DMTF specifications may be revised from time to time, the particular version and release date should always be noted." [3].

CIM 3 is a “common definition of management information for systems, networks, applications and services” [4]. It is an Object-Oriented Model describing the elements and entities in a managed environment, and how they are related to each other. First it was designed to model computing systems in an enterprise environment. Now it is used for systems and network management.

This standard is also called the CIM Schema, comprising definitions for 16 different areas, such as core (the CIM core), user, system, policy and security. Starting point is the core specification (current version 2.18 as part of the latest CIM version), and within this core the base class is ManagedElement (see Figure 25). The specifications are published as Microsoft Visio graphics or PDFs. There is no publication available for any UML or other object-oriented modelling tool. Please note that the published Visio graphics are UML-like and that the CIM language (Managed Object Format, MOF) contains a property definition for UMLPackagePath 4 and most CIM MOF specifications have this property applied. Furthermore, CIM allows specifying classes and their instances in the same specification.

All CIM specifications are available in MOF 5, but unfortunately there is no tool support from the DMTF for non-members. Some open source tools and products are available on the market. Companies such as Sun and Microsoft have incorporated CIM specifications into the management of their software systems (such as Microsoft’s WMI for their windows platform).

The abstract root class ManagedElement defines different associations 6 with itself (see Figure 25). to describe common properties such as LocalIdentity and ConcreteIdentity. The next level of classes inherits all these specifications. Here one can find definitions for Capabilities, Configuration, Location, Product, Settings and MethodParameters. All associations in CIM can be characterized as standard association or weak reference. Similar, aggregations can be standard or weak. Furthermore, one can define a Composition Aggregation. Classes within the CIM specifications can be typed ‘experimental’ or ‘deprecated’ [5].

The most interesting class in the second level of the hierarchy is ManagedSystemElement (see Figure 25). This class allows to define physical elements (such as devices) as well as logical elements (such as enabled logical elements, jobs, software identities and

---

3 The latest version of CIM is version 2.19.1 from October 8th, 2008.
4UMLPackagePath qualifier was introduced to define Package hierarchy with DMTFCR01592 / ARCHCR00037 and has been applied to the CIM Schema in 2.13.
5 This is DMTF MOF and should not be mistaken for OMG MOF
6 Some of the ManagedElement recursive associations: ConcreteIdentity, LocalIdentity, ConcreteComponent, Dependency, ConcreteDependency and HostedDependency and two aggregations (Component, OrderedComponent, RelatedElementCausingError, Synchronised and Spared.
LogicalElement), in essence, physical resources are PhysicalElements, while logical resources are LogicalElements (to be exact, most of them are EnabledLogicalElements). The latter group includes System, LogicalDevice, Service and ServiceAccessPoint. All other CIM specifications use this classification hierarchy. To give an example, the CIM network specifications defines a number of Collections (ManagedElements) for IP addresses, BGP group and logical elements (EnabledLogicalElement) for Service (i.e. VLAN Service), ServiceAccessPoint (i.e. VLAN SAP including DHCP / DNS / OSPF configurations), System (i.e. hosting VLAN services, including administrative domains and computing systems) and LogicalPort (as LogicalDevice).

Virtualization is introduced to the CIM core specification via virtual system management capabilities and virtual system management service. Both classes are experimental in CIM version 2.19.1. However, the starting point for information on CIM virtualization is the Virtualization Management (VMAN) Initiative [6], announced by the DMTF on November 27th 2007 and on September 16th 2008. VMAN is based on the Open Virtualization Format (OVF), a specification submitted by a group of DMTF member companies. [7] The OVF comprises a number of specification capturing profiles (system virtualization profile and virtual system profile), devices (generic device resource virtualization profile) and allocation (resource allocation profile and allocation capabilities profile). VMware has published a whitepaper on the OVF [8] and SourceForge hosts an open source project for OVF [9]. The DMTF has published OWV as DMTF specification DSP0243 and the OVF profiles as separated documents. A general overview of DMTF published documents can be found here [10].

The underlying concept of virtualization in the DMTF-CIM is to enable management of system virtualization, the latter one being the substitution process creating virtual resources. This class of resources is (usually) based on physical resources, but might (or will) expose different parameters and qualities [11]. The concept is captured by the System Virtualization Model, which identifies a host computer system and one (or more) virtual computer systems, including their devices, configurations and resource allocations (states). The relevant specifications are published in a number of profiles, each of which contains classes, properties, methods and values. The two abstract profiles are the Resource Allocation Profile and the Allocation Capabilities Profile. Resource pools and resource allocations are the key elements for the virtualization model.

![Figure 1 - CIM Virtualization, Resource Pools and Allocation](image)

ResourcePools are logical entities provided by the host system, intended to provide mechanisms to allocate and assign resources. A Virtual Resource (LogicalDevice, including state) can be linked to a Host Resource (LogicalDevice, including state) via the HostPool (ResourcePool). Please note that

---

this relationship between virtual resource and the host should only be used for relatively static connections (for example disk drives), and not for highly dynamic relationships (for example memory or processor resources).

A virtual system can be in one of the following four states: defined, active, paused, suspended. A newly created system starts in the state defined, which can be compared to a power-off state of a computer. Once activated, the system can be used. It might transit to the state paused (light sleep) or suspended (deep sleep).

Virtual devices need to be presented by a LogicalDevice subclass. CIM provides a number of profiles for virtual devices, such as CPU and memory.

CIM also provides a generic profile for the virtualization of devices, which can be applied if no specific profile has been published by the DMTF for a particular device.

**Summary**

The CIM model was designed to manage desktop computing environments. During its evolution, the CIM morphed into a Data Model that allows specifying distributed computing environments. Hence, the CIM model has a system view which impacts the definition of its entities. The network concepts in CIM have to be understood following this system view which can be difficult to network experts.

CIM consists of two parts: the Visio UML-like graphical representation of the model and the DMTF-MOF textual specifications. Within DMTF-MOF, classes and instances are specified and the name spaces for both are defined. Thus, the CIM model is a blend of an Information Model and a Data Model. Although as a modelling principles, conceptual modelling is based on ordering our knowledge of a domain in an abstraction levels for a better understanding of the domain and coping with the complexity. The mix of conceptual entities and their instances in CIM results on difficulties to understand the conceptual semantics independently of instantiation details which can be related to a specific platform/implementation.

The structure of the current version of CIM (2.19.1) focuses on a system, distributing specifications for network elements and services across three main parts: service, service access point and system. Logical device is used to specify concrete network elements, focusing on the capabilities of a network interface card (such as ports). The CIM does not provide specifications for important network concepts, such as separation of physical and logical device interfaces, templates for network element classes (such as routers, bridges) and models for Quality of Service.

### 2.3 The Shared Information and Data Model, SID

The Shared Information and Data Model (SID) is standardised by the TeleManagement Forum (TM Forum) in [12] and is starting to be standardised in the ITU-T in SG4. The SID is part of the NGOSS programme and thus closely linked to the TM Forum's architecture and business modelling (eTOM). As of today, the SID provides business and system view definitions for designing and managing a telecommunications network.

The SID defines domains and entities on several levels of detail. It enables the design of services and network resources in conjunction with products and customers, thus providing the necessary associations to link all resources to business activities. For example, services are categorised as customer or resource facing, where the former are services such as a VPN that a customer is directly aware of, whilst the latter are internal network services, such as MPLS or BGP, that the customer is unaware of. The SID can also be used as a toolkit that allows modellers to select what they need to model specific applications. Most of the technical specifications of the model are contained in their own domain-specific addendum, such as the document GB922-1POL [10] for the policy framework. However, the SID is limited with regard to modelling autonomic environments in
four main areas. First, the SID policy framework is inflexible compared to that of DEN-ng; see [11] [17] [18] for more details. Second, the business-to-network translation (and vice versa) was not realised in any tangible form in the SID. Third, important concepts such as context are completely missing from the SID. Finally, the SID does not provide artefact for the design of the entities behaviour such as state machines.

The main modelling domains in the SID are Customer, Product, Resource, Service and Common Managed. The latter is concerned with modelling entities that are common to all the other domains, such as Policies. Figure 2 illustrates how these domains interact from an abstract view. This view is centred on the BusinessInteractionItem to illustrate that the SID is more aligned with the requirements of representing business related concepts than network management concepts.

Therefore, the SID is not favourable for the application of building the Autol Information Model due to the highlighted issues. However, there are commonalities between the SID and the next reviewed information model, DEN-ng, and we intend to maintain compatibility between the extensions proposed in Autol and these two information model.

### 2.4 Directory Enabled Network-new generation, DEN-ng

The DEN-ng [17] was conceived as the foundation to building an Intelligent Network. It adds a business view to the model by providing business concepts to the network and the system ones. Also, DEN-ng is primarily concerned with the management of policy from a customer right the way through to the configuration of network and system devices [17] [20].

Autol has chosen DEN-ng because of three key properties. First, it is the only model that was built from the beginning to support roles and patterns (note that the SID picked this up, as it was in part based on an early version of DEN-ng) this provides inherent extensibility. Second, it uses a number of powerful abstractions [20] [22] that dramatically simplify management. Third, it is the only model whose design philosophy was to support orchestration of services through their entire lifecycle. Most models are limited to modelling the current state of an entity; DEN-ng models the lifecycle of an entity using state machines. DEN-ng is currently being adopted as the standard information model within the technical work of the ACF.

The SID used an earlier (v3.5) version of DEN-ng, but then the vision of DEN-ng and the SID diverged. The current version of DEN-ng, v6.6.2, and has been extensively remodelled.
and extended in many areas compared to the version used in the SID. Its main advantage is that it provides coherent multiple viewpoints of a communications network. Business goals are used to govern everything, from an individual managed entity to a domain of entities. DEN-ng was used to build FOCALE [26], an autonomic networking architecture that defines a novel set of control loops that analyse data, determine the current state of the entity being managed, compare the current state to the desired state, and reconfigure the managed entity to set its state back to the desired state. This is done through the application of context-aware policies to realise business goals.

The DEN-ng model applies patterns in a very strict way, enabling formal transformations through detailed semantic definitions [22]. Similar to the SID, DEN-ng is designed as a toolkit, making it possible to derive domain-specific information and data models by means of selecting an appropriate set of features and extending them for a given domain.
3 Autol Information Model, AIM

This section describes the Autol Information Model (AIM). The AIM, being at the core of the Autol project, specifies a set of abstractions defining entities of managed communication networks following the requirements and constraints defined in [1]. The artefacts defined in the model are independent of vendor, platform and repository. Thus, the AIM represents a conceptual model of the network management information and their structure in the Autoi context.

From the AIM model, a set of data models will be derived that will enable the concepts to be mapped to specific data models standards and technologies. A data model represents the state of the network entities to which the data relate to. The derivation of data models from a single common information model enables data coherency, understanding and interoperability across heterogeneous environments. The data models will be derived in direct relevance to the use cases demonstrated by WP6 and described in D6.2 [13].

This section provides the AIM model specifications, which are based on a subset of the DEN-ng information, version 6.6.2 (cf. section 2.4). The first part (section 3.1) introduces and describes the model specifications by discussing the structure and core concepts. The second part (section 3.2) provides instantiation examples of different types of information from the AIM model to illustrate the usability of the AIM in designing concrete examples.

3.1 Model Specifications

The specifications of the AIM model are presented as:

- Class diagrams with entities (classes) and their relationships (associations) in UML\(^8\) [14];
- Semantics, as textual definitions of entities and their semantics;
- Constraints of entities and/or their relationships in OCL [15], or textual.

The specifications in AIM are structured into domains, with each domain relating to a specific aspect of the Autol managed communication network. The AIM domains relate directly to UML packages, i.e. the specification of the domain Service are in the UML package Service. The following major domains have been identified:

- Root - The Root domain represents the top level entities defining the common properties to all entities in the Autol environment.
- Resource - The Resource domain represents the entities supporting services in the networks both physical as well as logical.
- Service - The Service domain consists of a set of layered entities that are used to manage the definition, development, and operational aspects of Services provided by the Autol system.
- Management - The Management domain consists of a set of layered entities that can be used in managing the behaviour and definition of entities in other Domains. This domain is a collection of the following domains which take part in the management functions in the managed communication networks (cf. Figure 28):
  - Policy - The Policy domain consists of a set of entities that define specifications (e.g., templates) and definitions of Policy entities that can be used in managing the behaviour and definition of entities in other Domains.

\(^8\) Unified Modelling Language
- **Context** - The context domain consists on a collection of entities that are used for the definition data that result from the set of all interrelated conditions in which an entity exists.

- **MetaData** - The MetaData domain consists of a set of concepts defining structured data that describe characteristics of the managed entities to aid in the identification, discovery, and management of the described entities.

The AIM domains dependencies are described in section 7.3.1. This section also describes other packages part of the AIM which supports the management functions.

The level of abstraction of the AIM model covers requirement **REQ-INF-MODEL-01** (cf. section 7.2).

### 3.1.1 AIM Design Process

The following process has been applied to migrate from the DEN-ng to the AIM.

- **Step 1**: extract a subset of DEN-ng based on the **AutoI** modelling requirements (cf. section 7.2) resulting in specifications of the core of the AIM.

- **Step 2**: refine this extracted model, to remodel entities and their relationships in correlation with the **AutoI** architecture (concepts and constraints) as defined in D6.1 [1].

- **Step 3**: extend the resulting model with new artefacts to capture virtualisation and (resource and service) lifecycle.

### 3.1.2 Root Entities

The start of every class hierarchy in a model is a distinct root entity. Figure 3 shows the **RootEntity** of the AIM with its common properties to name, describe and identify of all (i.e. managed and unmanaged) entities of the AIM.

![Figure 3 - Root Entities Overview](image-url)

An **Entity** is an abstract class that extends the **RootEntity** class in order to represent classes of objects that have a specific function in the communication system domain. **Entities** can be either managed or unmanaged. More formally, an **Entity** represents objects...
that have a separate and distinct existence. They are not just a collection of attributes or an abstraction of behaviour.

*ManagedEntity* is an abstract base class that is used to represent entities in a managed environment that have the following common semantics:

- Each *ManagedEntity* is associated with at least one *ManagementDomain* (cf. section 7.3.2),
- Each *ManagedEntity* is governed by at least one *ManagementApplication* (cf. section 7.3.2),
- A fundamental characteristic of a *ManagedEntity* is that it can perform and/or respond to management functions. This distinguishes *ManagedEntities* from *UnManagedEntities*. The three principal types of *ManagedEntities* are *Resource*, *Service*, *Product* and *Context*.

An *Entity* semantics can be augmented by a specific type of data: *MetaData*. Thus the *MetaData* specifies data that describes, but does not contribute to or impact, the state of the *Entity* that the *MetaData* is applied to (more details in section 3.1.5.3).

An instance of a *RootEntity* (any of its subtypes thus any AIM instance) can have one or more identities. Each of these identities has to be unique in the namespace that the object instance exists in to enable the system to unambiguously refer to a specific instance from different instances of the same *entity*. Also, an identity is defined by a specific Identification method. *Identity* is a *ManagedEntity* (more details in section 7.3.4).

The AIM supports the definition of multiple identities for an instance of any entity in the model to guarantee the possibility to refer to the same instance via different identification mechanisms implemented by different networks systems. To give a short example, an instantiation of a management system might have object instances accessible via a middleware platform (using the middleware platform specific identification methods) as well as via Web Services, using Web Service specific identification and store information in a distributed repository (such as LDAP). Now, every object (instance) could have up to three identities: one within the middleware, one as Web Service and one for LDAP, or any combination of the three. It is important that every object has to have at least one unique identifier in order to be accessible.

3.1.3 Resources

The resource concepts defined in the AIM cover requirement *REQ-INF-MOD-RES-01, REQ-INF-MOD-RES-02* and *REQ-INF-MOD-RES-03* (cf. section 7.2).

The *Resource* entities are a collection of elements in the networks, physical as well as logical, supporting and delivering services. These resource entities are defined as managed entities.

Management of resources involves planning, configuration, and monitoring to capture performance, usage, and security information. This also includes the ability to reconfigure Resources in order to fine tune performance, respond to faults, and correct operational deficiencies in the infrastructure.
Figure 4 represents the top level concepts in the Resource domain. A Resource is the abstract base class for all entities that are inherently manageable and support the delivery of a Service.

A Resource can have an associated ResourceState to describe an observable behaviour of the resource for a finite period of time. The ResourceState can be part of a resource life cycle defined by a finite state machine.

Resource is defined by two types NonVirtualResource and VirtualResource.

The resource model, extracted from DEN-ng version 6.2.2, does not support virtualisation aspects. Thus, for the AIM, we did classify the resource concepts as VirtualResource or NonVirtualResource regarding if the resource is supported by virtualisation techniques as described by D1.1 [16].

3.1.3.1 NonVirtualResource

The resource entities described in this section are mainly defined by the DEN-ng resource model [17]. For a better understanding of the resource model, it is important to understand the modelling principal of abstraction applied to the network resource by dividing it into its physical and logical aspect to simplify the complexity of such an entity. The separation is simple to determine: if you can pick up the entity, then it is a physical entity (e.g. chassis and network cards); if not, it is a logical entity (e.g. protocols). The advantage of this separation is that it enables the modelling of each aspect to proceed in parallel. Otherwise, the entire model would have to change each time a new feature was added. This couldn’t scale.

As shown by Figure 4, the NonVirtualResource is a subtype of Resource. It defines the physical and logical entities of the networks where no virtualisation mechanism is supported.

At the Physical network level, the concept of PhysicalResource is captured as an abstract base class for describing different types of hardware and physical devices. Another definition relates to the physical identity of the physical resource, i.e. a resource has a physical identity if it has a physical manifestation that enables it to be held and have a label attached to it [Str04].

In the AutoI domain, the focus is on the PhysicalDevice, PhysicalLink and PhysicalPort concepts to support the specific artefacts defined by the Virtualisation plan architecture in D1.1 [16] and the management functions in D4.1 [18].
The *PhysicalDevice* is an abstract base class for representing hardware devices that can be managed. This class represents a convenient aggregation point for combining different aspects of a device (e.g., the cables, connectors, cards, power supplies, and other objects that together make up the device). Thus, it enables the device itself to have a physical manifestation (e.g., the "Internet Gateway Router" can be identified as a *PhysicalDevice*). Examples of this class include routers and switches, computers, and other end-devices that are managed. Also, the *PhysicalDevice* can be defined as a collection of Hardware which defines any type of hardware element as an atomic unit (cf 7.3.5).

The *PhysicalLink* is a concrete class that represents the connecting or cabling together of hardware entities. This enables both wireless and connector-based communication to be modeled.

The *PhysicalPort* represents an actual or potential end point of a topological (physical) link, and corresponds directly to a physical port on a topology map. *PhysicalPorts* are always contained by another physical object - they can't exist by themselves. The two most common examples are *PhysicalPorts* on a *Card* and on a *Chassis*.

The *LogicalResource* represents one or more logical aspects of a resource. It is modelled as an abstract base class for all logical resources that are inherently manageable. It describe any network entity that has no physical presence, examples of logical resources include protocols, file systems and memory caches.

A *LogicalResource* has to be supported by a *PhysicalResource*. This is defined by the aggregation *LogicalPhysicalResource*, Figure 4, as a holder to detail the characteristics of this support.

The main subclasses of *LogicalResources* (cf. Figure 34) are:

- *LogicalDevice* is an abstract base class for representing logical concepts and services that can be managed, but which are associated with the device as a whole. Conceptually, this represents the "brains" of the device. Also, it enables the device itself to have a single logical manifestation compare to its physical manifestation as a *PhysicalDevice*.

- *DeviceInterface* is a concrete class that represents the (logical) interface of a device. This is not a transmission entity; rather it is used to program a logical connection from a device to a network medium. Different types of DeviceInterfaces exist for the different types of network media (e.g., IP vs. ATM) that are used in a network to enable such media to be programmed. The combination of a *LogicalDevice* and a *DeviceInterface* is what a developer programs use to define Services that run on the device.

- *Connection* is a sub type of *ManagedTransmissionEntity* (cf. section 7.3.6). It is responsible for the transparent transfer of information between the *DeviceInterfaces* which represent logical termination points. A *Connection* may be either uni- or bi-directional.

- *Protocol* is an abstract base class for representing protocols that can be managed. Also, it can be defined as a formal set of rules and conventions that governs how two entities exchange information (usually over one or more types of network media).

The *LogicalDevice* is the entity that is responsible for creating a *DeviceInterface*. Since a DeviceInterface is a logical concept, it must be hosted by a physical entity. The physical entity that hosts DeviceInterfaces is a *PhysicalPort*. 
3.1.3.2 VirtualResource

As specified in the AutoI requirements [1], the virtualisation plan (VP) provides virtual resources to support services. Based on the VP architecture [16], the virtualisation concepts are captured in the AIM regardless of the technical details of the virtual system implementation. The AIM provides an abstract view of the virtualisation plane resources with respect to the management and orchestration plan.

Figure 5 - VirtualResource Overview

Figure 5 shows the VirtualResource in relation to the NonVirtualResources and Service entities. The VirtualResource represents the emulation of a physical or logical resource based on virtualisation mechanism/technologies. The underlying physical or logical characteristics of the underlying resources are hidden from their user, and a set of characteristics derived from the (hidden) underlying resources are instead presented to the user via the VirtualResource. Also, a VirtualResource is always hosted by at least one PhysicalResource.

A VirtualResource can have a specific VirtualResourceState associated based on its life cycle.

Please note that in the model, there are two types of VirtualResource:

- VirtualLogicalResource representing a virtualisation of a LogicalResource, example the java virtual machine is an emulation of a logical machine.

The principal sub classes of the VirtualResource are the VirtualLogicalDevice and VirtualRouter directly related to the Virtual Plane conceptual elements.
The **VirtualLogicalDevice** has the same semantics as a **LogicalDevice**, i.e. it defines the logical part hosted by the **PhysicalDevice**, but it is supported by a virtualisation mechanisms. From an architecture view, the VirtualLogicalDevice represents the component entity defined in the VP [16]. The VirtualLogicalDevice available resources (CPU cycles, RAM, disk space) are estimated taking in consideration the virtualisation overhead [16], the suitable values are available as attributes: availableRAM, availableClockCycle, availableHD.

Network connectivity of the VirtualLogicalDevice depends on the **PhysicalPorts** hosted by the PhysicalDevice.

The **VirtualRouter** is an emulation of a physical router i.e. it provides the routing functionalities. The VirtualLogicalDevice can host many VirtualRouters running in parallel. Each VirtualRouter gets allocated resources (assignedRAM, assignedHD, assignedClockCycle) as defined by the OVF file (formatted virtual machine).

The VirtualRouter can use one or an aggregation of logical connections defined by DeviceInterface. The DeviceInterface represents in this case the virtual link.

### 3.1.4 Services

This section relates directly to the Service Modelling requirements presented in Deliverable 6.1[2]. The service concepts defined in the AIM cover requirement **REQ-INF-MODEL-03**, **REQ-INF-MOD-SERV-01**, **REQ-INF-MOD-SERV-02**, **REQ-INF-MOD-SERV-03** and **REQ-INF-MOD-RES-03** (cf. section 7.2).

Service entities are elements that represent services which the network can provide the user while also representing the requirements of the user, i.e. what the user is asking of the network. The deployment of services and their interdependency with the availability of resources forms a large domain within the Autol solution. Modelling the information relating to the management of these services including their relationship with resources deploying the services is vital in realising a multi domain, platform independent system.

In Autol, the discussion is about the provision of two types of service:

- **Application Service**: refers to the service being offered to the end-user, such as voice, video, and Internet browsing. This is referred to as a **CustomerFacingService** in Autol.
- **Network Service**: refers to services which network resources provide, such as mobility, QoS, and security. This is referred to as a `ResourceFacingService` in AutoI.

Like resources, services are managed entities. The objective in managing services is to ensure that user requirements are met while using available resources as efficiently as possible.

The AIM model presents concepts from the domains of service, resource and management. In this way, a service to be deployed can be defined in a common language\(^9\) based on the information model (cf. section 4.1.3). This service definition can then be mapped to resource requirements which are defined in the same generic language. The information model ties the two domains together and allows both services and resources to be managed to reflect this relationship i.e. changes in service definition will impact resource allocation and changes in resource availability will affect service configuration (or requirements).

### 3.1.4.1 Basic Service Concepts

This section details some of the main service model concepts. A service is represented as an abstract `Service` class in DEN-\(ng\) with many inherited classes that more accurately relate to individual services.

![Diagram of Basic Service Concepts](image)

**Figure 7 - Basic Service Concepts**

\(^9\) The common language is generated from the information model plus the ontologies. This refers to the definition of the DSLs.
**Service** is the base class for the service hierarchy. A **Service** represents the object that will be instantiated. The service is described with specific parameters – fileName, author, threadRef [19]. Appropriate parameters are also inherited from its parent class **RootEntity**, these include commonName, keywords and version.

**CustomerFacingService** is a subclass of **Service** and is an abstraction that defines the characteristics and behaviour of a particular service as seen by the customer. In Autol, it equates to Application Service [13].

**ResourceFacingService** is a subclass of **Service**. It is an abstraction that defines the characteristics and behaviour of a particular service that is not directly seen or purchased by the customer. **ResourceFacingServices** are "internal" services that are required to support a **CustomerFacingService**. In Autol, it equates to Network Service.

**ServiceState** represents a unique set of information, valid during a particular time period during the life of a service.

**LogicalLocation** represents the location where the service is hosted in the network.

**ServicePackage** refers to the bundling of one or more **CustomerFacingService** to meet the requirements of a specified **Product**.

**ServiceBundle** refers to the bundling of a set of **ResourceFacingService** to meet the requirements of a specified **ServicePackage**.

**ResourceFacingServiceAtomic/Composite** refer to the Atomic/Composite pattern used throughout DEN-ng. **ResourceFacingServiceAtomic** refers to a stand-alone service which meets the requirements of a corresponding **CustomerFacingService**. **ResourceFacingServiceComposite** corresponds to a "grouping" or integrated set of **ResourceFacingServices** that collectively meet the needs of the **CustomerFacingService**.

**CustomerFacingServiceAtomic** and **CustomerFacingServiceComposite** apply the “Atomic/Composite” pattern used throughout DEN-ng. **CustomerFacingServiceAtomic** refers to a stand-alone service which meets the requirements of a corresponding business level product. **CustomerFacingServiceComposite** corresponds to a "grouping" or integrated set of **CustomerFacingServices** that collectively meet the needs of the business-level product.

### 3.1.4.2 Service Lifecycle Model Concept

The Autol service lifecycle follows a defined state machine as detailed in Chapter 3 of Deliverable 5.1 [19]. The model below extends the existing state machine model to deal more specifically with Autol service lifecycle [19]. The classes **ServiceState** and **ServiceEvent** represent the behaviour of the lifecycle.

**ServiceState** is a representation for a collection of unique information relating to the service, valid during a particular time period during the life of a service.

**ServiceEvent** represents a type of an event occurrence which causes the service state to change or transition to another state, section 3.1.5.5 gives more details regarding the Autol state machine model.
3.1.4.3 Service and Resource Relationship

The definition of the service-resource relationship forms the basis for self-management of the services and resources based on changes in either's (service and resource) context. Figure 9 illustrates the high-level service and resource relationships within the Autol domain.

**Figure 9 - Service Resource Relationships**

*ResourceFacingService* as its name suggests is the service aspect which forms the main relationship with the underlying resources. It is related to *LogicalResources* and *PhysicalResources* through two aggregations, *LogicalResourcesImplementRFS* and...
PhysicalResourcesHostRFS. LogicalResourcesImplementRFS species the required LogicalResources needed to operate the corresponding ResourceFacingService. The logical resources might refer to memory or device interfaces. PhysicalResourcesHostRFS species the required PhysicalResources needed to host the corresponding ResourceFacingService. The physical resources might refer to physical routers or ports. The aggregation VResourceSupportsRFS introduces the relationship between service and the virtual resource. VirtualResource “supports” the service in that it may be used to either host or implement the resources needed to allow the service function correctly.

3.1.5 Management Concepts

The management concepts presented in this section cover REQ-INF-MODEL-01, REQ-INF-MOD-MAN-02, REQ-INF-MOD-MAN-04 and REQ-INF-MOD-MAN-05 (cf. section 7.2).

3.1.5.1 Policy Domain

The Policy information model for AutoI (AIM) is a subset of the model defined in the DEN-ng 6.6.2. The concepts identified as relevant to AutoI are outlined in this section. There are two main types of concepts defined in the AIM with respect to policy based management. Firstly, those concepts that are used to describe the components of individual policy rules and how they relate to the managed entities defined in the rest of the AIM. Secondly, those concepts that describe the use of policies in managing sets of managed entities.

The AIM is primarily concerned with describing the structure and relationships of ManagedEntities from the perspective of AutoI. The Policy related concepts of the AIM are concerned with describing a management methodology that is flexible enough to be used for the management of all defined ManagedEntities. Interestingly, Policies are themselves ManagedEntities and so can be managed by other policies. This decision yields a highly flexible management methodology for use in the management of autonomic communications networks.

3.1.5.1.1 The Basics

The Policy model describes the different types of policies and concepts related to policies that can be defined using the AIM model. It describes the components of policies, and how they can be brought together to make individual policies.

The main abstract concept is that of PolicyConcept that is used to define attributes, methods and relationships common to any concept related to policies. The main subclasses of PolicyConcept are PolicyRuleStructure, PolicyRuleComponentStructure, PolicyCategory, PolicySubject and PolicyTarget.

3.1.5.2 PolicyRuleStructure

The PolicyRuleStructure concept represents the different forms that policy rules can take. For example, the ECAPolicyRule is a triple of policy components: an event clause, a condition clause and an action clause. This is a typical representation of policies and is compatible with most existing policy models including the SID (cf. section 2.3), Ponder, IETF and PDL (Policy Description Language). The UtilityFunctionPolicyRule is a policy rule that embeds a utility function within its condition. It is essentially a special form of condition–action policy rule, where the action chosen depends on the output of the utility function. This type of policy rule is highly flexible and useful when many variables need to taken into consideration for a decision to be made. Another type of policy is the GoalPolicyRule, that specific a desired goal in the form of desired preferences, but does not state how this goal should be achieved. GoalPolicyRule may be used to represent high-level goals that managed entities should aim to achieve, such as service levels or quality of service targets. For AutoI there are only these three PolicyRuleStructures. The PolicyRuleStructure can be used to define the behaviour of other ManagedEntities. For
example, the PolicyRuleStructure can be associated to the ContractualAgreement concept in the AIM in order to describe the appropriate type of agreement that should be used during negotiations. The PolicyRuleStructure can be associated to ContextData to define the appropriate level of detail that should be contained in that type of context.

Each PolicyRuleStructure has associated to it aStateMachine that represents the different states that the PolicyRuleStructure can be in. Essentially, the behaviour produced by the PolicyRuleStructure may be changed as network conditions change. This method of behaviour enables that policy to have deterministic behaviour under differing network conditions. The PolicyRuleStructure also has an association to PolicyRuleMetaData that defines some more information about the execution semantics of the PolicyRuleStructure.

![Figure 10 - PolicyRuleStructure](image)

3.1.5.1.3 PolicyRuleComponentStructure

The PolicyRuleComponentStructure describes the different types of components a PolicyRuleStructure is composed of. The ECAPolicyRule is composed of PolicyActionStructure, PolicyConditionStructure, and PolicyEventStructure. This enables us to define new types of components that can be embedded in a policy rule, and it enables us to describe different types of events, conditions and actions.

The ECAPolicyRule is the most popular form of policy type, and will be the most prominent in Autol. Its execution semantics are, on the occurrence of a set of events, if a set of conditions are satisfied, then execute the set of actions. The PolicyEvent component of the ECAPolicyRule is a hierarchical container of events that can occur simultaneously, in a specific order, or mutually exclusively. This makes the evaluation of policy rules scalable, as each policy condition will not need to be continuously evaluated as network conditions change. The PolicyCondition component of the policy rule can be used to evaluate thresholds as well as other Boolean expressions, including time based expressions. The values that can be evaluated within a PolicyCondition can be linked to any individual ManagedEntity or the triggering PolicyEvents. The PolicyAction component of the policy is related to changing or maintaining the state of managed entities. It can be an invoked method of the ManagedEntity or a link to a code / configuration operation so that entities unaware that they are being managed can be integrated into the system.
Now that there is a common representation of a policy rule, it can be used in many different ways; this is represented in the AIM through the use of categories. PolicyCategory in the AIM is an abstract class that defines how a policy rule should be used. In the AIM the main type of policy rule is the ManagementPolicy. The ManagementPolicy defines policies that are used to manage a system and can be associated to different types of PolicyRuleStructures. This means that the ManagementPolicy can be associated to ECAPolicyRules or UtilityFunctionPolicyRules. The PolicySubject concept represents the ManagedEntities that have their behaviour managed by policies, whereas the PolicyTarget concept represents the ManagedEntities that are being acted upon as the result of a policy decision. Both concepts are associated to the ManagementPolicy. ManagementPolicyMetaData and PolicyRuleMetaData represent the concept that data can be associated to individual policies to better define their
behaviour.

![Diagram of ManagementPolicy](image)

**Figure 12 - ManagementPolicy**

The main types of *ManagementPolicies* defined are *DeonticPolicies*. These policies relate to the concepts of authorisation, prohibition, obligation and exemption. These policies can be subject enforced or target enforced. By separating the enforcement entities, strong distributed management can be achieved, where entities can decide to enforce policies locally or depend on remote policies to manage their behaviour.
Other policy categories defined in the AIM are OrchestrationPolicy, subclasses of this are NegotiationPolicy, FederationPolicy and DistributionPolicy. The AMSPolicy category defines policies dedicated to the management of ManagementPolicies for a specific domain. The AMSPolicy can be used for example to manage access to a set of ManagementPolicies given the current Context of the AMS.

Figure 13 - PolicyCategory

Figure 14 - AIM PolicyCategories
3.1.5.1.5 Policy Applications

The Policy application model is defined to describe how policies can be used to manage a domain of managed entities. The ManagementApplication is any application that is used to manage a set of ManagedEntities constrained to a PolicyDomain.

![Policy Application Diagram]

Figure 15 - ManagementApplication

The PolicyDomain concept is used to define a group of ManagedEntities whose behaviour is defined by a common set of policies. This differs from a ManagementDomain which is more associated to a group of ManagementEntities that are owned by an organisation. The current PolicyApplications are the PolicyDecisionApplication (PDA), PolicyEnforcementApplication, PolicyConflictApplication and PolicyBrokerApplication. Note that the term Application is used in this model as opposed to the Point terms used to define the policy architecture by the IETF. This is to highlight the need for the systems to be distributed applications, instead of features of applications.

The PolicyDecisionApplication is an application that can be used to evaluate policies for other ManagedEntities that do not have the capability of making decisions. The PolicyEnforcementApplication is an application that can carry out the instructions of satisfied policies if the ManagedEntities cannot carry out the PolicyActions by themselves. The PolicyEnforcementApplication also monitors and verifies that the PolicyActions have had the desired effect. As multiple policies may be defined and deployed simultaneously, there is a very high probability that they may instruct conflicting actions to be performed in the network. The PolicyConflictApplication analyses the deployment of policies and the
runtime execution of policies to ensure that the desired behaviour is actualised in the network. The PolicyBrokerApplication is used to aid in the distribution of PolicyApplications. It can communicate between two or more PolicyApplications when decisions or actions need to be distributed. This is a concept that is missing from traditional PBM architectures. It is an essential component in Autol as it ensures that all PolicyApplications can be distributed.

![ManagementApplicationComponent](image)

**Figure 16 - ManagementApplicationComponent**

PolicyApplications can be composed of multiple PolicyApplicationComponents. Therefore, PolicyApplicationComponents can be shared between PolicyApplications. The components defined for Autol are the PolicyRepository, PolicyDistributionComponent, PolicyReceptionComponent, PolicyProxyComponent, PolicyAdministrationComponent, and PolicyEditingComponent.

The PolicyRepository is a component of a PolicyApplication which is used to store ManagementPolicies. It can be distributed or centralised, also it can be used to hold parts of policies, such as PolicyEvents, PolicyConditions and PolicyActions. The PolicyDistributionComponent is used by PolicyApplications to send information related to policies. This can be full PolicyRuleStructures, or individual PolicyRuleComponentStructure. The PolicyReceptionComponent is used by PolicyApplications to receive information related to policies. The PolicyProxyComponent is used by PolicyApplications to transform data in different formats into data conforming to that specified in the information model. It can also be used for example, to transform PolicyActions into CLI or shell scripts that must be executed to carry out the intentions of the PolicyAction. The PolicyAdministrationComponent is an interface to a PolicyApplication so that it can be managed. The PolicyEditingComponent is an interface to a PolicyApplication so that PolicyConcepts can be edited offline or online.

### 3.1.5.1.6 Autol Policy Continuum

Considering that the Autol architecture is not focused on a single administrative domain, there needs to be a slight modification to the concepts of the levels of policies for the Autol Policy Continuum (APC). The processes and methods developed to realise the policy
continuum are outlined in D4.1 [18]. This section is concerned with describing the aspects of the information model that support the policy continuum.

Business Level Policies / High Level Policies

These policies represent the high-level objectives and goals of a single organisation, or administrative domain. Therefore, there may be many high-level policies in existence in Autol that will remain local to a single domain, or shared across multiple domains. High level policies can be related to services that the organisation offers, or is willing to host for other domains.

The concept of domain in Autol is defined to be the scope of a single AMS and the scope of multiple federated AMSs. High level policies are used to describe high-level directives such as objectives, goals, priorities and preferences that are applicable to the whole domain. These directives are used to govern the decisions made at lower levels of the policy continuum.

For example, a high level policy may define an objective such as maximise resource utilisation, this objective can be used to influence a low-level policy that is used to decide whether to accept a new service deployment. The service should be accepted if its acceptance will maximise resource utilisation. However, typically there are multiple objectives that should be considered.

High level policies can be represented using Goal Policies, UtilityFunction Policies or ECA Policies. Concepts in the information model relevant here are those concepts that are used to help describe objectives. For example, if security is an objective then the meta-data of a managed entity describing its security features are relevant. Concepts typically related to high-level policies are used to help define the set of managed resources and services from the perspective of the whole domain. Other form of objectives are to minimise cost or maximise reliability, therefore these concepts are also represented in the information model using meta-data.

Orchestration Level Policies

Orchestration policies are related to high level policies, as the decisions outlined by high level policies impact the decisions being made by orchestration policies. These policies control the behaviour of negotiations between domains, the distribution of tasks and information between domains, and the agreements for federation between domains. They are also used to make decisions on a domain wide basis, for example, should a federation be created or not.

For example, orchestration policies are put in place to govern the decisions made during a federation activity. If a particular AMS has a high-level objective to maximise its service reliability, and the activity of federation would meet this objective then it would be willing to participate in the federation. Orchestration policies can be represented using ECA policies and Deontic policies. To ensure that high-level level policies are continuously being evaluated, there is typically a component nested within the orchestration policy constraining its decisions to lie within the directives of the associated high-level policies.

The concepts in the information model that can be used to help define orchestration policies are those that describe the different types of services and resources that are relevant to inter-domain processes such as negotiation and federation.

System Level Policies

These policies are related to the management of an autonomic management system (AMS), which is administrative over a set of virtual entities. An AMS has a two tier management architecture, where the AMS can be managed using policies, and individual virtual entities in the AMS can be managed using policies. Therefore there are nested control loops at work at the system level. This is manifested through the use of policies that manage policies. These policies are associated to high-level policies in that the decisions
that can be made at this level are directly impacted by the decision made at the high level. These policies must be more adaptive than those policies at other levels, as they must be able to react to changes in the state of the managed system, and must continuously meet the directives set out by high-level policies.

It is envisaged that there will be dedicated policy authors working at this level that are capable of defining policies that can operate in the given context. The novel concepts defined in the information model to facilitate this functionality are, context, state and metadata. Therefore, policies can be tied to the state of the managed system and they can also be tied to the context of the managed system. Metadata associated to the AMS as a whole and to individual VRs can be used to aid in decision making.

For example, a particular VR can have a policy associated to it that is adaptive to the changing conditions of the network, and is also cognisant of the business policies of the AMS. This policy may decide depending on the state of the AMS to prioritise more critical services in times when the network is experiencing congestion. When the congestion levels recede, then it may reconsider its priorities and accept new service types. The aim of these policies is to help exhibit self-* behaviour from the AMS.

**Component Level Policies**

Component policies are defined to manage individual Components as defined in the Autol architecture. The policies can be associated to managing virtual resources and services deployed onto the component. As the AMS is dedicated to managing the virtual routers residing on the Components, policies for defining the behaviour of the Components are required. The manageability of Components in Autol is limited to the functions defined within the vCPI and the vSPI\[16\]. These policies are also governed by business policies.

An example usage of Component level policies would be to automatically expand the capabilities' of the routers in an AMS if the AMS was experiencing a certain level of congestion.

**Instance Level Policies**

Instance level policies are embedded inside devices that can perform their own decision functions. An example of this is routing functionality where the decisions need to be performed on a per packet basis. Policies defined at a higher level may make decision to re-configure devices capable of hosting instance level policies. In Autol, these policies are concerned with the control of real-time algorithms. For example, an instance policy maybe defined to describe the weights associated to links in the BGP protocol, or the intervals associated to discovering best available wireless network in a mobility environment. These policies can be represented using ECA policies

3.1.5.2 Context

DEN-ng defines context as “The Context of an Entity is a collection of measured and inferred knowledge that describe the state and environment in which an Entity exists or has existed. In particular, our definition emphasizes two types of knowledge – facts (which can be measured) and inferred data, which results from machine learning and reasoning processes applied to past and current context. It also includes context history, so that current decisions based on context may benefit from past decisions, as well as observation of how the environment has changed.”\[20\]

The core concepts of the context model, see Figure 17, are derived from the DEN-ng context model in AIM.
Any ManagedEntity can have Context or ContextData associated with it.

The **Context** of a ManagedEntity is a collection of knowledge and data that result from the set of all interrelated conditions in which an Entity exists. Context can have multiple distinct sets of related data and knowledge that are used to adjust its state in accordance with the changes in the environment that it exists in. It is represented as a set of aggregations to **ContextData**, where **ContextData** is a class that focuses on one specific type of data and/or knowledge that is aggregated by the Entity’s Context. The semantics of this aggregation are represented by the **ContextDataDetails** association class.

ContextData is an auxiliary class that is only used when Context contains multiple distinct types of different data that need to be combined in order to determine the overall context of an entity.

The **ContextDataFact** and **ContextDataInference** are used to represent additional data that is either known a priori or can be inferred from other knowledge about a given ContextData.
3.1.5.2.1 Context aware policy model

The HelpSelectsPoliciesToActivate aggregation defines a set of Policies that should be loaded and activated based on the current ContextData. Hence, as context changes, policy can change accordingly, enabling the system to adapt to changing demands. Note that this selection is an intelligent decision, in that the selection process depends on other components that are part of a particular context. Another intelligent decision is the PolicyResultAffectsContext association, which enables policy results to influence Context via the ContextControllerComponent, the application that manages Context. For example, if a policy execution fails, not only did the desired state change not occur, but the context may have changed as well.

The selected working set of Policies uses the ContextAwarePolicyEnablesManagedEntityRoles association to define and enable the appropriate ManagedEntity roles that are influenced by this Context; each ManagedEntityRole defines functionality that the ManagedEntity can use. In this way, policy indirectly (through the use of roles) controls the functionality of the system, again as a function of context. Similarly, ContextAwarePolicyEnablesMgmtInfo defines the set of management data that is useful for this Context; ManagementInfoAffectsContext represents feedback from these management data regarding the execution of the policy rule. Once the management information is defined, then the two associations MgmtInfoAffectsContext and ManagedEntityRoleAffectsContext codify these dependencies (e.g., context defines the management information to monitor, and the values of these management data affect context, respectively). Finally, the ContextControllerComponent defines its own set of ManagedEntityRoles and ManagementInfo to use to monitor the environment; feedback from the ManagedEntities identified by their roles and specific measurements (in the form of ManagementInfo) are used by the ContextControllerComponent to operate its own finite state machine (FSM). This FSM is used to orchestrate the actions of the ContextControllerComponent, including which ManagedEntities it should determine the context for, how a particular element of context should be analyzed, and what procedures for determining its context should be used [20].
3.1.5.3 **MetaData**

Self-knowledge is the ability of a system, service or managed entity to have knowledge about itself. This is required so that the entity can make informed decisions that are cognisant of its roles, capabilities and capacities, and indeed the roles, capabilities and capacities of managed entities it is in contact with. For this reason, the AIM has concepts to represent the roles, capabilities and capacities of managed entities. It is this self-knowledge that can be used to influence the decision making processes developed in AutoI.

*Capacity, Capability and Role* are sub types of MetaData. As defined in section 3.1.2, each Entity can have extra-semantics represented via the MetaData providing knowledge to the entity itself and to the other entities.

The MetaData concepts are investigated as potential information to be used by the management functions currently defined by the management plan work in WP4 and by the use cases demonstration work in WP6. As a modelling task, the refinement of these concepts is still open following the outcomes and discussions.

3.1.5.3.1 **Roles**

Role based access control [21] is a methodology used to separate the entity requesting access to a resources, from the resource. Therefore, the access control system can define a set of roles that can access the resource rather then explicitly specifying the entities that can access the resource. Entities that are eligible to take on these roles can then access the resource. The assignment of roles is flexible and is typically application specific. Roles make the access control process scalable as the roles are defined independent of the number of potentially accessing entities.

However, roles do not just have a benefit with respect to access control; in fact, the scalability of roles has been seen as an attractive choice in many management systems. Policy-based management systems, for example, make extensive use or roles to describe the requesting entities and the target entity of a policy. This dual separation yield a PBMS that can define and enforce policies independent of the number and types of subject or target management entities in the system.

Assigning roles to a managed entity in the AIM can occur in two ways. The role can specify a set of criteria that the managed entity must meet before it can be assigned the role. This may include authentication, or holding a specific token. The role can be assigned explicitly to a managed entity.

The AIM models roles as a subclass of MetaData that can be associated to any ManagedEntity. ManagedEntities can take on many roles and roles can be shared among many ManagedEntities. There are a number of explicit roles of significance defined in the AIM. Such as FirewallRole, RouterRole and VPNLogicalDeviceRole. These roles are used to indicate the roles of LogicalDevices and can be used to determine the multiple roles of a single LogicalDevice with respect to configuring it.

Roles can be defined for CustomerFacingServices and ResourceFacingServices and help in the automated composition of services. Note that this is an open discussion with WP5 to establish if this will be used.

3.1.5.3.2 **Capabilities**

In order to ascertain the features, resources and services that a ManagedEntity is capable of supporting, we need a notion of Capabilities. With this notion, a management system can use it to aid in the prediction or planning of service deployment. Capabilities can be assigned to ManagedEntities as a method of pre-computing if it can be used to sustain a
particular set of features or services. By pre-computing these values, more efficient and effective management decisions can be made. The types of capabilities that need to be computed are determined by the ManagedEntity.

Capabilities in the AIM are subclasses of the MetaData class. Therefore, Capabilities can be associated to any ManagedEntity in the AIM. It is envisaged that capabilities will be used as part of the decision making processes in the Autol architecture, specifically for planning and prediction, as capabilities are pre-computed in some cases. Capabilities can also be computed dynamically depending on the frequency of change of the data.

### 3.1.5.3.3 Capacities

Some attributes of managed entities can be better described by way of capacities. Capacities are thresholds or ranges that give a measure to aspects of a ManagedEntity. A ManagedEntity may need to have a specific capacity of memory for example before it can commit to activating a service. Capacity is not a realtime measure; it defines the maximum or minimum threshold of a quantity of a ManagedEntity. In the AIM, Capacities are modelled as subclasses of MetaData and so can be associated to any ManagedEntity, in much the same way a Role or a Capability can.

It is envisaged that capacities will also play a part in decision making in Autol, in specific when multiple services rely on common Capacities, a priority ordering will need to take place so that the Capacity is not breached. Capacities can also be used as part of UtilityFunctions to indicate which ManagedEntities have higher utility with respect to a management function.

### 3.1.5.4 Topology

As mentioned in D1.1, the virtual plan will provide information about the (physical/logical/virtual) resources topology to support management and/or orchestration functions [16].

The AIM support the definition of topology based on graph representation. Figure 19 shows an overview of the topology concept.
A Topology refers to a representation of a collection of connected ManagedEntities. This representation is based on a Graph that defines a collection of a set of Nodes and a set of Edges (cf. section 7.3.7).

In the context of AutoI, the Nodes refer to ManagedEntities like PhysicalResource, VirtualResource or Service. The Edges can refer to Connection.

Topology is defined as a subtype of ManagedEntity which implies that Topology can be managed. At this stage of the project, Topology is viewed as inferred information from the available data. The inference of a Topology is based on rules defined by policies. Typically, a rule represented by a PolicyRuleStructure (cf. 3.1.5.1.2) determines the connection between specific types of Nodes. This is modeled, Figure 20, as two associations PolicyDetermineNodeSourceEdges and PolicyDetermineNodeTargetEdges expressing how the details of the relationship between Edges and Nodes is defined.
The AIM model provides state machine artefacts supporting the design of the behaviour of entities such as Service and Resource. As a first stage in the project, the behaviour describes the life cycle of a ManagedEntity from a management point view like the life cycle of the deployment of a Service (cf. section 3.1.4.2) and the management of a VirtualResource [16].

A StateMachine is an abstract class designing behaviour model composed of a collection of States, collection of Transition between those States and Events triggering a change of state. Figure 21 shows the modelling of StateMachine details.

The management of ManagedEntities can be based on their behaviour described by a StateMachine model. A ManagedEntity can be in a certain state, its transition to a new state is defined by specific rules. These rules can be specified using policies.

At this stage of the project, the management and orchestration tasks are investigating how to use the state machines to support the management of some entities. As a modelling task, the refinement of these concepts is still open following the outcomes and discussions on this matter.

### 3.2 Instantiation Examples

This section describes objects instantiation examples of different types of information from the AIM model. These instantiation examples illustrate the usability of the AIM in designing concrete examples.
3.2.1 Resources Instantiation

For the resource concepts instantiation, a simple example is used: A company has one main office and one branch office. Communication between the two offices is usually unencrypted. A new company policy specifies that data tagged as “classified” must not leave the company network unencrypted. Therefore, when such data is sent between the two offices, setup of an encrypted channel is necessary. In order to have the network set up, the first step is to provide a virtual layer where the border routers of both office networks get a new virtual image, providing virtual routers that will tunnel all data through a preconfigured encrypted link.

Each border routers are hosted by a physical machine. Figure 22 shows the resources instantiation. Component1 and Component2 are VirtualLogicalDevice instances hosted by the two machines instances of PhysicalDevice class.

3.2.2 Services Instantiation

For the service concepts instantiation, a simple example is used for this illustration: A customer requests the creation of a secure virtual private network (VPN). The model in Figure 23 illustrates the initial service modelling instantiation based on the concepts of the AIM model. The depicted service model breaks the required service up into manageable service aspects i.e. CustomerFacingService and ResourceFacingService that can be eventually translated to resource requirements.
The service being requested by the customer is an aggregation (or composition) of individual (atomic) customer services therefore the overall service, SecureVPNService is a CustomerFacingServiceComposite which would be composed of 2 x CustomerFacingServiceAtomic, Security and VPN. In turn, the CustomerFacingServiceAtomics require (Association:RFSRequiredbyCFS) a set(composition) of ResourceFacingServices each to meet their security and VPN mandates. This relationship is represented by the two classes: SecureService (Security) and TunnelService (VPN). Each of these services might be met through IPSec and IP tunnel respectively – this would be bring us to the next stage of the modelling process i.e. the definition of the Domain Specific Language(DSL), which is handled in Activity 3.3. Similar to the CustomerFacingServices, each of the ResourceFacingServiceComposites is composed of individual ResourceFacingServiceAtomics. SecureService would require a set of appropriate NetworkSecurityForwardingServices, for example: Encryption algorithm used to encrypt the flow of traffic or Authentication algorithm to specify the type of authentication applied to the traffic flow [RFC4301]. In turn, the TunnelService requires some means (i.e. protocol) for encapsulating and transmitting the data in the tunnel, EncapsulationService. Again, specific protocols would be the next layer of work i.e. the DSL.
4 Usage of the Autol Information Model

4.1 Model Driven Process

The usage of Information Models is linked to the flexibility of the model-based processes that depend on it. Model-based processes are those that take as input some aspects of an Information Model and perform some processing to produce an output, which may be executable, skeleton code, data base schemes, domain specific languages or analysis tools. For Autol, the model-based process shall be used at a minimum to generate data model schemas, libraries that can be used to parse and serialise data for transfer and a set of Domain Specific Languages (DSLs) that can be used to create domain specific solutions in languages which are understood by domain experts.

Process overview
The process is presented as a set of steps:

- **IDENTIFICATION:**
  The information model is refined and specialised using UML to suit the requirements of the system(s) being developed. The information model is stereotyped to indicate those classes, attributes, methods and associations that are of interest. Note multiple stereotypes can be assigned to the same UML entities.

- **TRANSFORMATION:**
  The information model is then passed into a transformation engine to produce an EMF\textsuperscript{10} based information model. This model format forms the base of many modelling technologies. The information model can be used to generate multiple data models, given sufficient transformation logic.

- **GENERATION:**
  The EMF model can be used to generate a Java based library that can be used to generate and parse XML data conformant to the model. Tools to encourage the rapid integration of the EMF model into the components of Autol are generated.

4.1.1 Process step: IDENTIFICATION

The Identification is typically carried out by a domain expert in collaboration with an information modelling expert. The domain expert will define the scope of the information model that is relevant to their needs, and the information modeller will ensure that those needs are catered for within the model. This may involve specialising some aspects of the information model. For example, say the domain expert is interested in an information model to describe the attributes and relationships associated to a virtual router. The domain expert may describe that they require the virtual router to depend on physical resources, with special capabilities of hosting a virtual machine monitor. The domain expert may also describe that these virtual routers should be able to make connections to other routers, either real or virtual. The information model architect ensures that the needs are met by extending the information model in the most appropriate fashion.

As the domain expert is not totally concerned with end-user or customer services, this aspect of the information model is deemed irrelevant and scoped out. The remaining entities (UML) are then stereotyped using the convention imposed by the information

\textsuperscript{10} Eclipse Modelling Framework
modeller. This stereotyped portion of the information model may then be used to create a new model which is a specialised subset of the original information model. At this stage in the process, there exists a model to be further used to aid the domain expert.

4.1.2 Process step: TRANSFORMATION

To make some use of the defined model, it can be transformed into a form that can be more readily used towards tool or code generation. Transformation entails changing the representation of the information model into another format, where some transformation logic is typically required. For example, to transform the information model from an UML formatted model to an EMF formatted model, some transformation logic is required. EMF is a simpler language than that of UML and there may be some loss in precision that has to be catered for. The transformation logic can be defined by the information model expert or there may be pre-defined transformations. These are so called model-to-model transformations.

Once in the desired model format, more specialised tools can be used to support code or tools generation that will make use of the information model. EMF and AndroMDA [23] are two of the prominent tools available that support model driven development techniques.

4.1.3 Process step: GENERATION

Different domain experts put different requirements on the use of the information model for their needs. This step is very much dependent on those needs. In fact this is the most flexible step, as there exists many supporting tools (facilitated by EMF and AndroMDA) to support generations. The generation step takes as input a tailored model, and produces code compliant to a generation template. Some logic may be introduced into the template so that the generated code more accurately reflects the needs of the domain expert. For example, if the domain expert is interested in a tool to help them model the requirements and characteristics of virtual resources and virtual networks in the form of a language (xTEXT [24]) or GUI (GMF [25]) that is tailored to his needs, this can be automatically generated for them. The generated tool can include code completion for the language interface and constraint checking facilities for the GUI.

At a minimum there will be a data model generated to support the transfer of data between different components in the AutoI architecture. There will also be support tools to aid in the creation of data in the correct format, and the parsing of data. There will also be support tools to aid in the translation of this data to a format understood by third part components; however, these tools will not be available until month 18.

A Current Realisation

Figure 24 illustrates a current realisation of the model based process where the information model is used to instantiate a data model that conforms to an EMF based XML schema. Therefore the data that can be understood between the components in AutoI is in this format. This type of tool generation is readily supported with the modelling tools of Eclipse and has been developed to illustrate a proof of concept.
The AIM (Autol Information Model) is transformed into the EMF modelling format using our model-based process. The tools available in Eclipse to manipulate the EMF model are used to generate some jars (Java Archives) that have expose APIs to generate data conformant to the data model. This data can then be stored in the Context Information Service, and may be retrieved by other components such as the Service Enablers Platform. Third party components will need to translate their data into formats compliant with that understood by the Autol components.

**Future Realisations**

Future realisations of the Model based process include the use of DSLs and Ontologies to enhance knowledge representation, and decision making capabilities in Autol.

A DSL is an abstraction mechanism which handles the complexity in a given domain by providing a customised language that represents concepts and rules specific to that application domain, rather than using the terminology of a generic programming language. This enables developers to work directly with domain concepts, and hence model the system more accurately and in a way that is more easily understandable. For example, a Service DSL can be used by a service enabler expert to define and describe the invariant characteristics and behaviour of a new network service. A DSL’s advantage is that the concepts and relationships of the information model can be used to enhance syntax checking, code completion, and semantic analysis.

The ontologies will be used to aid in the reasoning and understanding of data and information; they can also be used to aid in the integration of third party components into Autol.

**4.2 Information part of the Autol Interfaces**

The AIM model provides structured information representing types of data that is exchanged between the components of the Autol planes described in D6.1 [1] and D4.1 [18].
The interaction between the AutoI planes is defined by external interfaces provided by the components of each plan. An interface represents a method signature with parameters (inputs and outputs). These parameters represent the exchanged data that is captured and structured by the AIM model. This modelling process is part of the identification step described on the section above.

The second step is the definition of these interfaces using a common formalism based on the set of DSLs. This will provide common interface representation. The Ontologies and DSLs are under creation in Task 3.2 & 3.3 in year 2 of the project.

As an illustration of the usage of the AIM, the three operations, part of the vCPI interface, are described in relation to the information model.

Each component in the virtualisation plane provides the vCPI interface as described in D1.1 [16]. As an illustration, the three functions instantiateVM(), changeVMState() and instantiateLink() are described in relation to their parameters as data instantiated from the AIM.

- The instantiateVM() function allows the creation of a virtual router instance, i.e. a running instance in the physical machine and also an object instance of type VirtualRouter describing the properties of the running virtual router to the management plan.

  VirtualRouter instantiateVM(URLLocation ovfFile)

- The changeVMState() function allows changing the state of a specific virtual router instance. This instance can be defined by the id or by the instance itself:

  Void changeVMState(Identity vrId, VirtualResource state)

  or

  Void changeVMState(VirtualRouter vr, VirtualResourceState state)

- The instantiateLink() method sets up a new link between two Virtual Routers. The link is defined by an instance of type Connection.

  Connection instantiateLink(VirtualRouter vr1, VirtualRouter vr2)
5 Conclusion

This document provides the specifications of the Autol Information Model (AIM). The model was developed as part of the activities in work package 3 and is the interoperability core of the Autol project. The bases of the AIM model are the Autol modelling requirements described in D6.1 and it is aimed at the Information Networking of Future Internet. The main part of the AIM model was extracted from the DEN-ng model version 6.6.2.

The first step was an analysis of the three major candidates for information models against the Autol modelling requirements (section 2). DEN-ng was identified as the most suitable model. The next step was to define the design process to build the AIM, based on the specific Autol requirements state-of-the-art in model design and development. Section 3 describes the resulting modelling process. This section also defines, specifies and explains the core concepts of the AIM, including instantiation examples which should help the reader to gain a good understanding of the conceptual entities of the model. The third step, in Section 4, shows how to use the AIM as the core artefact provides a common language ensuring a common understanding across the Autol planes and components. The usability of the AIM is based on the integration of the model with the model driven process.

The AIM itself captures all necessary entities needed for and concerned with the management of communication networks.

At the current design and development stage, there are some parts of the AIM which will need further work in year 2 of the project to complete the design. In particular:

- The Policy Model originating from the DEN-ng needs to be aligned with results from WP4, as soon as they are available.
- The Context Model originating from the DEN-ng needs to be aligned with results from WP4, as soon as they are available.

Furthermore, the AIM design makes the following assumptions, which will need proof through their usage within the Autol project:

- The designed Topology Model supports the definition of topologies based on available data, each data being an instance of the AIM.
- The management of the behaviour of service and resource entities is based on state machines. That means the behaviour of an entity is described by a state machine process which is governed by rules (policies) which in turn are defined by the management plane.

In addition, in the year 2 of the project the AIM is planed for use as follows:

- Data Models
  From the AIM model, a set of data models will be derived that will enable the concepts to be mapped to specific standards and technologies. A data model represents the state of the network entities to which the data relate to. The derivation of data models from a single common information model enables data coherency, understanding and interoperability across heterogeneous environments. The data models will be derived in direct relevance to the use cases demonstrated by WP6 and described in D6.2 [13] and the design of the Information and Service Platform of D4.1 [18]. This work is part of task 3.3 in WP3.

- Autol Ontologies
  The AIM Model is used to define facts, and ontologies are used to augment facts with additional semantics that cannot be represented using the Unified Modelling Language.
The set of ontologies will be used to extend model knowledge with semantics to enable reasoning. This work is part of task 3.2 in WP3.

- **Autol DSLs**

Domain Specific Languages (DSL) are used to address the specific system tasks in an interoperable manner. Each DSL is a special subset of the common language (the AIM model and the Autol Ontologies) designed to be of specific use to a constituency of a network administrator. For example, a Service DSL can be used by a service enabler expert to define and describe the invariant characteristics and behaviour of a new network service. The DSLs relates to the specific domains within Autol: Service, Resource, Management (Policy). This work is part of task 3.2 in WP3.
6 References


7 Annex

7.1 CIM Specification Resources

![Diagram of CIM Core Specifications (Overview)](image1)

![Diagram of CIM Core Network Specification (Overview)](image2)
7.2 Modelling Requirements

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>REQ-INF-MODEL-01</td>
<td>Abstraction Provides a set of information abstractions that enable services that require network resources to express their needs to the network in a form that the network can understand. In the same way, network resources are enabled to be defined in such a way that the services can understand.</td>
</tr>
<tr>
<td>REQ-INF-MODEL-02</td>
<td>Management Interface Provision of suitable Autol management concepts and appropriate relationships that enable the services and resources to be governed.</td>
</tr>
<tr>
<td>REQ-INF-MODEL-03</td>
<td>Service Interface Provision of suitable Autol service concepts and appropriate relationships that enable the services to be deployed or configured based on business requirements and resource availability.</td>
</tr>
<tr>
<td>REQ-INF-MODEL-04</td>
<td>Resource Interface Provision of suitable Autol resource (virtual) concepts and appropriate relationships that enable resources to self-organise based on service requirements.</td>
</tr>
<tr>
<td>REQ-INF-MODEL-05</td>
<td>Behaviour Provision of relevant concepts/relationships which enable the effective representation of behaviour (orchestration interface) required in the Autol network.</td>
</tr>
<tr>
<td>REQ-INF-MODEL-06</td>
<td>Semantic Interoperability Augment information abstractions (concepts) with additional elements that enable a greater understanding of the semantics of these abstractions via the use of ontologies.</td>
</tr>
<tr>
<td>REQ-INF-MODEL-07</td>
<td>Dynamic Updating of Information Due to the autonomic nature of the Autol environment, it is likely that the information may change as the network evolves due to changes in environment and context.</td>
</tr>
<tr>
<td>REQ-INF-MODEL-08</td>
<td>Domain Specific Languages Provide a set of Domain Specific Languages, derived from the Information model, that are specific to the domains of service, resource and management while being constrained from the common Autol information model. Each entity will speak its own language while recognising the overall language of Autol (information model).</td>
</tr>
<tr>
<td>REQ-INF-MODEL-09</td>
<td>Information Structure Provision of an information schema derived from the overall information model that can be used in the storage and retrieval of appropriate information across the distributed architecture.</td>
</tr>
<tr>
<td>REQ-INF-MOD-SERV-01</td>
<td>Service Abstraction Provides sufficient abstractions to allow services to represent their requirements to the management functionality, which then transforms those requirements into a particular virtualisation of some or all the network resources.</td>
</tr>
<tr>
<td>REQ-INF-MOD-SERV-02</td>
<td>Network Service representation A network service refers to a resource-facing service which the network would provide, for example: mobility, QoS. It will be possible to describe a network service, its interface northbound towards an appropriate application service and its interface southbound for provisioning and monitoring of resources.</td>
</tr>
<tr>
<td>REQ-INF-MOD-SERV-03</td>
<td>Application Service representation An application service refers to the consumer-facing service being offered to the end-user. It will be possible to describe an application service such that high-level business and end-user goals can be translated to low-level resource requirements.</td>
</tr>
<tr>
<td>REQ-INF-MOD-RES-01</td>
<td>Resource Abstraction</td>
</tr>
<tr>
<td>--------------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>Provide sufficient abstractions to allow the definition of virtual resources in terms of high-level business requirements</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>REQ-INF-MOD-RES-02</th>
<th>Physical/Logical resource relationships</th>
</tr>
</thead>
<tbody>
<tr>
<td>Provide sufficient abstractions to enable management of virtual resources towards their associated logical and physical resources.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>REQ-INF-MOD-RES-03</th>
<th>Service (Network/Application) relationships</th>
</tr>
</thead>
<tbody>
<tr>
<td>Provide sufficient abstractions to enable management of virtual resources towards their requested network and application service</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>REQ-INF-MOD-RES-04</th>
<th>System virtualisation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supports operations to define system virtualisation based on classical hypervisors</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>REQ-INF-MOD-RES-05</th>
<th>Virtualisation behaviour</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supports concepts of virtualisation behaviour such as aggregation and splitting of virtual resources</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>REQ-INF-MOD-RES-06</th>
<th>Virtual Resource configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supports operations to define the configuration of virtual network resources, in particular virtual routers and the envisioned virtual topology</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>REQ-INF-MOD-MAN-01</th>
<th>Resource Management</th>
</tr>
</thead>
<tbody>
<tr>
<td>Provides sufficient abstractions to support resource management (usage, availability, (de)composition, virtualisation and assurance)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>REQ-INF-MOD-MAN-02</th>
<th>Management plane</th>
</tr>
</thead>
<tbody>
<tr>
<td>Support for management functions when adapting to changes in resource or service context</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>REQ-INF-MOD-MAN-03</th>
<th>Sharing information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Support for the sharing and re-use of information across the self-management functions and also the orchestration and virtualisation operations</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>REQ-INF-MOD-MAN-04</th>
<th>Policy Models</th>
</tr>
</thead>
<tbody>
<tr>
<td>Support for a policy model that enables the self-* capabilities of the Autol framework using a policy continuum to provide translations between business, network, and implementation policies</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>REQ-INF-MOD-MAN-05</th>
<th>Policy Based Framework</th>
</tr>
</thead>
<tbody>
<tr>
<td>Support for a policy based framework that can detect and resolve conflicting goals between different management self-functions as well as between different planes</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>REQ-INF-MOD-MAN-06</th>
<th>Policy Language DSL.</th>
</tr>
</thead>
<tbody>
<tr>
<td>A set of domain specific languages (DSL) need to be developed to support the needs of different constituencies of users in Autol. A DSL is an elegant way of providing users with a method of building programs that address a well defined problem domain. Hence, multiple policy DSLs can be customised to suit the needs of multiple users</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>REQ-INF-MOD-CON-01</th>
<th>Contextual Abstraction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Provide sufficient abstractions to allow the representation of context changes in the following areas: (1) Resource (Network &amp; Service), (2) Service (Network &amp; Application), (3) Network (Virtual, Physical &amp; Logical) and (4) Business Requirements</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>REQ-INF-MOD-ORCH-01</th>
<th>Orchestration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Orchestration functionality requires concepts and relationships to describe dependencies among (re)configuration tasks, and the relationship and ordering of those tasks, in order to govern the self-* functions</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>REQ-INF-REP-01</th>
<th>Information sharing</th>
</tr>
</thead>
<tbody>
<tr>
<td>The system shall take advantage of the distributed (P2P) nature of the Autol network elements to share relevant network management data without the expense involved in maintaining a centralised server</td>
<td></td>
</tr>
</tbody>
</table>
7.3 AIM Specification Resources

7.3.1 AIM Domains dependencies

The elementary content of the Root domain is required for the definition of the content of the other domains in the AIM.

![Figure 27 - AIM packages and their dependencies Root Domain Usage](image1.png)

Figure 27 - AIM packages and their dependencies Root Domain Usage

Figure 28 shows the constituencies of the Management domain. Each sub domain supports a specific aspect of the management functions in the managed communication networks.

![Figure 28 - Management domain constituencies](image2.png)

Figure 28 - Management domain constituencies
The Management domain content applies to the managed entities defined by the Resource, Service, Product domains as shown in Figure 29.

Figure 29 - Management Domain usage

### 7.3.2 ManagedEntity Relationships to ManagementDomain and ManagementApplication

Based on the DEN-ng model, each ManagedEntity is associated to one or more ManagementDomain. This relationship is defined by the association class ScopeManagedEntitiesDetails (cf. Figure 30).

A ManagementDomain represents a special grouping of ManagedEntities that has two important properties:

- It is used to partition managed objects into a meaningful logical grouping. One important use of such a grouping is to provide a means to define which Element Management System ¹¹ (as well as which Network Manager System ¹² defined by the classical telecommunication management architecture) manages and monitors which set of devices. It also provides a means to show how management functions are distributed and scaled.

- It defines a common administrative domain that is used to administer the managed objects that it contains. This implies that all of the managed objects contained in this

---

¹¹ Element Management System (EMS) manages one or more of a specific type of telecommunication network element. Element Management is concerned with managing network elements on the Network Element Management Layer of the TMN (Telecommunications Management Network). For more details, refer to the ITU-T Recommendation series M.3000.

¹² Network Management System is part of the Network Management Layer of the TMN. This layer has the responsibility for the management of a network as supported by the Element Management Layer. For more details, refer to the ITU-T Recommendation series M.3000
ManagementDomain are administered similarly - either by the same user, group of users or policy.

From the Autol architecture perspective, the ManagementDomain will be defined by the Autonomous Management System (AMS) domain. The concept of AMS domain is started being defined in D4.1 [18]. The ManagementDomain semantics, in the AIM model, will be refined following this work on the definition of the AMS.

As defined in the DEN-ng model, each ManagedEntity is governed by at least one ManagementApplication or a PolicyApplication (subtype of ManagementApplication) since all ManagedEntities are governed using policy management. The PolicyApplication Governs ME Details (cf. Figure 30) association defines the set of PolicyApplications that are involved with processing policy decisions and performing policy actions on a set of ManagedEntities. Note that this association does NOT imply an action being taken on a ManagedEntity. Rather, it shows that a PolicyApplication is involved with applying policy to a particular ManagedEntity.

A ManagementApplication is a particular type of Application (which is a type of LogicalResource) that is used to manage one or more types of ManagedEntities. Management may be accomplished directly by this entity, or indirectly (via this entity directing other ManagementApplications to do its job).

A PolicyApplication is a type of ManagementApplication that provides Resources and Services for controlling the state of the system and objects within the system using policies. Control is implemented using a management model such as a finite state machine. It includes installing and deleting policy rules as well as monitoring system performance to ensure that the installed policies are working correctly.

The PolicyApplication semantics, in the AIM model, will be refined following the work on the definition of the AMS and the use of policies in WP4.

![Figure 30 - ManagedEntity Relationships to ManagementDomain and ManagementApplication](image)

### 7.3.3 ManagedEntity Subclasses

Error! Reference source not found. shows the ManagedEntity subclasses.
7.3.4 Identity and Identification

An instance of any entity of the AIM model has at least one unique Identity which is specified by IdentityIdentificationDetails based on the Identification mechanism (implementation and platform specific).

An Identity of any entity is a ManagedEntity.
7.3.5 **PhysicalResources**

Figure 32 shows the main subclasses of the *PhysicalResources*.

![PhysicalResources Diagram](attachment:image.png)

**Figure 33 - PhysicalResources Overview**

7.3.6 **LogicalResources**

Figure 33 shows the main subclasses of the *LogicalResources*.

![LogicalResources Diagram](attachment:image.png)

**Figure 34 - LogicalResources Overview**

*ManagedTransmissionEntity* is an abstract base class for describing different types of logical entities that are or help to form connections that transmit and/or receive information.
7.3.7  Graph entities

A Graph is an abstract class that represents a collection of a set of Nodes and a set of Edges that establish relationships between the Nodes.

A GraphComposite is a type of Graph that contains multiple GraphComposites and/or GraphAtomic instances. This enables hierarchies of graphs to be formed, where the GraphComposite object acts as a container, and the GraphAtomic object is the actual graph. This in turn enables different semantics (e.g., policies or access rules) to be applied to all objects contained within a GraphComposite.

A GraphAtomic represents an instantiable Graph that can stand on its own.

A Node is one of the two fundamental building blocks of a Graph. It is an abstraction of an element part of the collection of interconnected elements of the Graph.

An Edge is a connection between one or more Nodes in a Graph. Typically, a Graph is depicted in diagrammatic form as a set of dots for the Nodes, joined by lines or curves for the Edges.