# ANA Project

Autonomic Network Architecture


Sixth Framework Programme

Priority FP6-2004-IST-4

Situated and Autonomic Communications (SAC)

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

Deliverable D1.4/5/6_v1.1

# ANA Blueprint – First Version Updated

Version 1.1

# ANA Project

## Autonomic Network Architecture


| | |
|---|---|
| **Project Number** | FP6-IST-27489 |
| **Project Name** | ANA - Autonomic Network Architecture |
| **Document Number** | FP6-IST-27489/WP0/D1.4/5/6_v1.1 |
| **Document Title** | ANA Blueprint – First Version Updated |
| **Workpackage** | WP1 |
| **Editors** | Christophe Jelger (UBasel)<br>Stefan Schmid (NEC) |
| **Authors** | Christian Tschudin (UBasel)<br>Christophe Jelger (UBasel)<br>Ghazi Bouabene (UBasel)<br>Guy Leduc (ULg)<br>Lorenzo Peluso (FOKUS)<br>Manolis Sifalakis (ULancs)<br>Marcus Schoeller (ULancs)<br>Martin May (ETHZ)<br>Matti Siekkinen (UOslo)<br>Rudolf Roth (FOKUS)<br>Stefan Schmid (NEC)<br>Tanja Zseby (FOKUS)<br>Thomas Plagemann (UOslo)<br>Vera Goebel (UOslo) |
| **Reviewers** | David Hutchison (ULancs) |
| **Dissemination level** | Public |
| **Contractual delivery date** | 31st December 2007 |
| **Delivery Date** | 15th February 2008 |
| **Version** | 1.1 |

**Abstract:**

This document provides an update of the first version of the ANA Blueprint Architecture. It describes the initial reference model of the Autonomic Network Architecture (ANA) developed within the EU FP6 IST Project 27489.

The Blueprint includes a definition of the basic abstractions, core concepts and communication paradigms of ANA, defines abstract interfaces that an actual implementation of a network node must provide in order to support the ANA architectural concepts, and the information flow model for the relevant control and management information of an autonomic network.

**Keywords:**

ANA Blueprint Architecture, Basic Abstractions, Communication Paradigms, ANA Node, Information Flows

# Executive Summary

The goal of the ANA project is to explore novel ways of designing and building networks beyond legacy Internet technology. The ultimate goal is to design and develop a novel network architecture that can demonstrate the feasibility and properties of autonomic networking [1]. As specified in the description of work [2], it is the intension of the project to facilitate the self-* features of autonomic networking such as self-configuration, self-optimization, self-monitoring, self-management, self-repair, and self-protection by an innovative network architecture.

Because autonomic networking is very recent research area and there does not yet exist an autonomic network architecture, the ANA project is doing significant spadework on this wide and challenging research topic.

Designing a network architecture implies identifying and specifying the design principles of the system being developed. The architecture defines the atomic functions and entities that compose the network and specifies the interactions which occur between these various building blocks.

As the first draft of the ANA architectural blueprint specification, this document describes the initial reference model of the ANA autonomic network architecture. It includes a definition of the basic abstractions and concepts of the ANA architecture and the communication paradigms. The document also includes the abstract interfaces that an actual implementation of a network node must provide in order to support the ANA architectural concepts, and defines the information flow model for the relevant control and management information of an autonomic network.

**Revision History:**

| | | |
|---|---|---|
| 31.12.06 (M12) | Version 1.0 | First draft of the ANA Blueprint Document |
| 31.12.07 (M24) | Version 1.1 | Update of the ANA Blueprint - first version |
| | | Summary of changes:<br>• Update of the Introduction and Conclusions sections (section 1 and 7)<br>• Update of the Compartment and Node Compartment sections (section 3.1 and 3.2)<br>• Introduction of the Compartment API (section 3.3.3)<br>• Update of the ANA implementation description (section 5.4) |

# Table of Contents

# 1 INTRODUCTION

This document provides an update of the first version of the ANA Blueprint. It is based on the first version of the document, which was produced during the first 9 months (M4-M12) of architectural work in 2006 and used as the basis for the development of the initial ANA prototype.

This update of the ANA Blueprint reflects the lessons learnt from the ongoing prototyping efforts of the ANA core system during the second year of the project. As the development of many of the ANA communication system functions is still ongoing, a second and more mature Blueprint document will be released at the end of 2008 (M36), which will reflect all the necessary changes according to the experiences gained during the further development.

This first version of the Blueprint describes the initial reference model of the ANA autonomic network architecture. This includes a definition of the basic abstractions and concepts of the ANA architecture and the communication paradigms. The Blueprint also includes the abstract interfaces that an actual implementation of a network node must provide in order to support the ANA architectural concepts. Finally, the Blueprint defines the information flow model for the relevant control and management information of an autonomic network (e.g., for configuration, monitoring, routing, optimization, resilience) and the necessary interfaces.

Note: The ANA Blueprint document encompasses deliverables D1.4v1 (First ANA Blueprint specifications), D1.5v1 (Autonomic functional blocks), and D1.6v1 (Information flow data items and interface specifications) which together specify the core operation of ANA.

## 1.1 Motivation and Objectives

This section recaps the main drivers behind the Autonomic Network Architecture (ANA) and its high-level objectives.

The overall objective is to develop a novel network architecture and to populate it with the functionality needed to demonstrate the feasibility of autonomic networking. In order to avoid the same failings of network architectures developed in the past, the guiding principles behind the architectural work in ANA are achieving maximum *flexibility* and providing support for *functional scaling* by design.

The former indicates that the project members do not envision a "one-size-fits-all" network architecture for the various different types of networking scenarios (e.g., Internet, Sensor Networks, Mobile Ad hoc Networks, Peer-to-peer Networks). Instead, the aim of ANA is to provide an architectural framework that enables the co-existence and interworking between different network architectures.

Functional scaling means that a network is able to extend both horizontally (more functionality) as well as vertically (different ways of integrating abundant functionality). New functions must be integrated; otherwise we do not have scaling of functionality, only function accumulation. In the context of ANA, the ability of functional scaling is considered vital, as this provides the basis to allow dynamic integration of new, *autonomic* functionalities, as they are developed.

The Internet and the OSI model are examples of a functionally non-scalable approach: both are based on the premise of factoring out functionality. The "canonical set" of protocols collected in the Internet-Suite has done a good job so far; however the limitations of this one-size-fits-all approach have become visible. This is not surprising, since the networking layer of the Internet has not aimed at functional scaling by design: it has evolved through a 'patchwork' style. Additions were made in a stealth-like way and have not dared to change things. The main examples are the introduction of the hidden routing hierarchy with AS, CIDR or MPLS, as well as other less successful initiatives such as RSVP, multicast, mobile IP and MANET.

As a result, the objective of the ANA project is to provide an architectural framework – a "meta architecture" – that allows the accommodation of and interworking between the full range of networks, ranging from small scale Personal Area Networks, through (Mobile) Ad hoc Networks and special purpose networks such as Sensor Networks, to global scale networks, in particular the Internet.

An important step towards such a "meta architecture" is to design a framework that is able to host different types of network. As a result, ANA encompasses the concept of *Compartments* as a key abstraction that allows co-existence and interworking of different types of network through a minimum generic interface.

Furthermore, the operation of different types of compartments must be analysed in order to identify the fundamental building blocks of the abstraction. The decomposition of compartments into the core functions helps to understand how the necessary flexibility and functional scalability to provide *autonomicity* can be achieved through compartments.

An important concern in developing a new architecture is to ensure that networks have the necessary emergent properties to support the applications and users running on them.

## 1.2 Scope of Deliverable

The Blueprint documents focus on the overall architectural aspects of ANA. As such, they define the basic abstractions and building blocks of ANA and present their basic operation and interactions.

As it is most important to establish a sound architectural framework, the focus of the initial work was primarily on the overall architecture. How autonomicity can be achieved will be more carefully studied once the architectural framework becomes stable and the key building blocks and their basic operations have been identified.

Therefore, the blueprint does not capture the details of the ongoing development work or the results of other areas of work in the project, such as routing, service discovery, functional composition, monitoring, etc. The outcomes of those areas of work are describes in the following deliverables:

- D1.7: Prototype implementation of the information flow framework [3]
- D1.8: Prototype implementation of the core ANA software [4]
- D2.1: First Draft of Routing Design and Service Discovery [5]
- D2.2: Functional Composition Framework [6]
- D2.3: Initial Design and Evaluation of Inter-Compartment Communication Schemes [7]
- D2.4: Initial Design and Prototype Implementation of the Functional [8]
- D2.6: Design and Evaluation of Self-Association and Self-Organization mechanisms for Network Compartments [9]
- D2.7: Design, Prototype and Evaluation of an Customizable Overlay Compartment [10]
- D2.8 Service Discovery and Routing Schemes for intra- and inter-Compartment Service Provisioning [11]
- D3.1: Monitoring Framework [7]
- D3.2: Resilience/Security Framework [13]
- D3.3: The Monitoring Part of the ANA Architecture (v1) [14]
- D3.4: The Self-Optimization Part of the ANA Architecture (v1) [15]
- D3.5: Specification of failure detection and fault management in the ANA architecture [16]
- D3.6: The Resilience Part of the ANA Architecture (v1) [17]

## 1.3 Structure of the document

The structure of this document is as follows: Section 2 defines the ANA terminology. Section 3 describes the core abstractions of the ANA network architecture and the basic communication paradigms. Section 4 further outlines how autonomicity is achieved in ANA. Section 5 defines the ANA node, its basic components and operation as well as the abstract interfaces. Section 6 defines the information flow framework that is the basic concepts for information sharing in ANA. Finally, section 7 concludes the document.

# 2 TERMINOLOGY

## 2.1 Abbreviations

| | |
|---|---|
| ANA | Autonomic Network Architecture |
| API | Application Programming Interface |
| BP | Bootstrap Protocol |
| FB | Functional Block |
| IC | Information Channel |
| IDP | Information Dispatch Point |
| IDT | Information Dispatch Table |
| MC | MINMEX Controller |

## 2.2 Definitions

**Identifier:**

- An *Identifier* consists of a finite sequence of symbols of a given alphabet.
- Identifiers are used for identification of an entity within a set of entities (e.g., to single out one or more objects from a set of objects).
- Identifiers are typically persistent and globally unique within a given identifier space.
- Within ANA, identifiers are used to identify objects, resources, etc.

***Name:***

- A *Name* is a globally unique, persistent identifier used for recognition of an entity (e.g., object, resource, and host). For example, a name can be used to resolve the address/locator of the entity.
- Within ANA, names are used to identify an entity (e.g., object, resource, and host). Names can thus be used to resolve the identifier or address/locator of an entity, which is necessary to gain access or communicate with the entity.

***Address:***

- An *Address* is an identifier that is used for routing and forwarding.
- Addresses entail the information that is needed to transport data/information from a source to a destination.

- In case of structured addresses, which define the location of an entity within a topology, addresses are also called Locators.

**Label:**

- A *Label* is a node-local identifier for an Information Dispatch Point.

**Functional Block (FB):**

- A *FB* is an information processing functions in ANA, which for example generates, consumes, processes or forwards information.
- FBs run on one node only, which means a node has full control over the functional blocks it hosts.
- FBs can have zero or more input and output.

**Information Channel (IC):**

- An *IC* is the abstraction of a communication service that is provided by a (underlying) compartment or the underlying system.
- Conceptually ICs can be seen as the channel/medium over which communication between FBs takes place.
- ICs are provided by a compartment and only exist within compartment bounds.
- ICs can be logical, i.e. they can span several physical network devices and be viewed as distributed objects made of FBs and other, lower level ICs.

**Information Dispatch Point (IDP):**

- An *IDP* is the entity that allow decoupling for ICs and FBs. That is, access to an FB or IC is done via the IDP it is bound to.
- IDPs allow dynamic (re-)binding of FBs and ICs in a way that is transparent to the "client" of the FB or IC. That is, the entity bound to an IDP can be changed but the IDP used to interface with the entity remains.
- IDPs perform no processing except data forwarding (actual processing of data is performed by FBs).

**Compartment:**

- A *Compartment* is a policed set of FBs, IDPs and ICs, which enables communication for its members according to some commonly agreed set of communication principles, protocols and policies.
- A compartment's protocols, policies and communication principles form a sort of "recipe" that all compartment entities must obey.

- The common recipes are typically the communication principles, protocol(s) and policies to be used.
    - Examples of common communication principles: how naming, addressing, routing, etc. is handled
    - Examples of protocols: communication protocols between peer FBs on different nodes, etc.
    - Examples of common policies: membership (join/leave) procedures trust, etc.
- Compartments are the context in which FBs and ICs exist. This context includes related internal management activities.
- The FBs of a compartment communicate through Information Channels (ICs). ICs can be seen as abstractions of "connections" or "communication channels" through underlying compartments. In case there are no underlying compartments, ICs represent the underlying communication channels (outside the ANA world) that are used for transferring information between FBs (e.g., an inter-process communication channel, an L2 channel).
- Some compartments may require a minimal set of (composed) FBs in each and every participating ANA node. This is considered again as part of the compartment policies. Those composed FBs are closely related FBs interconnected through IDPs.

# 3 BASIC ABSTRACTIONS AND PARADIGMS

This section introduces the basic abstractions of ANA, namely the compartment concept, the network compartment and the node compartment.

## 3.1 Compartments

### *3.1.1 Background and Motivation*

Recent research has led to the conclusion that today's networks have to deal with a growing diversity of commercial, social and governmental interests that have led to increasingly conflicting requirements among the competing stakeholders. Clark et al. refer to this development as "tussles in cyberspace" [18]. A pragmatic way to resolve those tussles is to divide the network into different realms [20] or turfs [19] that isolate the conflicting or competing interests. In the context of ANA, the term compartment is used, which likewise captures the concepts of virtual networks and sub-networks.

The compartment abstraction, as one of fundamental concepts of ANA, allows atomization or decomposition of communication systems and networks into smaller and more easily manageable units. For example, compartments will allow decomposition of today's global IP network into appropriate sub-networks, which can be managed more autonomously from the overall network (e.g., a different addressing or routing scheme can be applied inside each compartment). Atomization or decomposition also helps reducing complexity by abstracting compartment internals to the outside world and hiding complexity of communication beyond the compartment from the individual communication elements.

In addition to the ability to separate concerns through atomization or decomposition, the compartment abstraction serves as the unit for the federation of compartments into global-scale communication systems and networks. For example, compartments allow interworking across heterogeneous network domains (e.g., nodes that are located in compartments that use different technologies can interwork through a common overlay compartment).

The fact that the compartment abstraction allows federation or composition of heterogeneous compartments also enables ANA to "encapsulate" today's networks; i.e. compartments offer a kind of "wrapper mechanism" that allows accommodation of legacy network technologies within ANA. Since support for backwards compatibility and interworking with already deployed networks is a key requirement for any new network architecture, some architectural compromises are typically made to achieve this. However, as the compartment abstraction allows hiding the internals of individual

network domains and enables interworking among heterogeneous compartments, ANA enables full integration of existing network technologies or domains in the ANA world.

### 3.1.2 Compartment Definition

Compartments implement the operational rules and administrative policies for a given communication context. The boundaries of a communication context, and hence the compartment boundaries, are based on technological and/or administrative boundaries. For example, compartment boundaries can be defined by a certain type of network technology (e.g., a specific wireless access network) or based on a particular protocol and/or addressing space (e.g., an IPv4 or and IPv6 network), but also based on a policy domain (e.g., a national health network that requires a highly secure boundary).

ANA anticipates that many compartments co-exist and that compartments are able to interwork on various levels. In a sense, compartment provides a "recipe" that specifies how the compartment operates; e.g., it defines how to join a compartment, who can join, and how the naming, addressing and routing is handled.

The complexity and details of the internal operation is left to each compartment. For example, registration with a compartment can range from complex trust-based mechanisms to simple registration schemes with a central database or a public DHT-based system; resolution of a communication peer can be handled implicitly by the compartment's naming and addressing scheme or require explicit actions (e.g., resolution of an identifier to a locator). It is important to note here that compartments have full autonomy on how to handle the compartment's internal communication – i.e. there are no global invariants that have to be implemented by all compartments or all communication elements.

A compartment is defined by the set of abstract entities (members) which are able and willing to communicate among each other according to compartment's operational and policy rules. Conceptually, a compartment maintains some form of implicit database which contains its members, that is, each entry in the database defines a member. Before one can send a data packet to a compartment member, a resolution step is required which returns a means to "address" the member. Note that the above definition does not specify whether a member is a node, a set of servers or a software module. This rather abstract definition of compartment membership permits to capture many different flavours of members and communication forms.

A second core building block of ANA is the functional block (FB). FBs are code instances and state that can process (send, receive, forward, etc.) information. Typically, the communicating entities of a compartment are represented in the ANA architecture through FBs. They implement the functionality that is required to communicate within the compartment. As such, the FBs can also be regarded as the processing elements or functions hosted by an ANA node that constitute the "compartment stack".

FBs can also be composed of a set of two or more FBs that are hosted on an ANA node. Further information on how composition of FBs is achieved on ANA nodes is provided in Deliverable D2.2 [6].

The fact that a FB can represent the whole range from an individual processing function (as an atomic FB) to a whole compartment stack or even a network node (as a composed FB), makes this abstraction very useful. It permits to capture many different flavours of communication entities and compartment types (for example, compartments of simple software modules or compartments of client hosts and servers).

Communication inside a compartment is mediated via Information Channels (ICs). ICs can be either of physical or logical nature. Examples of physical ICs are a cable, a radio medium, or memory. A logical (or virtual) IC can represent a chain of packet processing elements and further ICs. The IC abstraction captures various types of communication channels, ranging from point-to-point links or connections, over multicast or broadcast trees to special types of channels such as anycast or concast trees. Figure 3-1 illustrates a number of different types of ICs. Note that broadcast is considered a special case of multicast here.

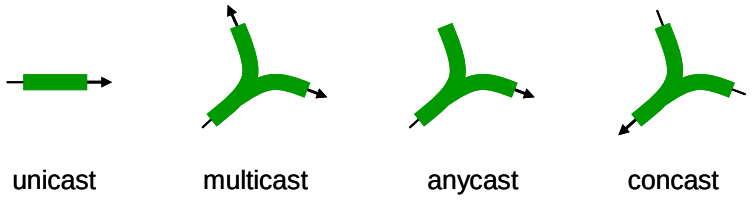



**Figure 3-1. Different Types of Information Channels (ICs)**

In order to connect flexibly to FBs and ICs, the ANA framework introduces Information Dispatch Points (IDPs). IDPs are bound to FBs or ICs, and hence provide a decoupled "entry point" (handle) to the FB or IC. This decoupling allows dynamic (re-)binding of an ANA entity (i.e. a FB or a IC) in a way that is transparent to the entities that communicate with the FB or IC. For example, a FB A that sends information to another FB X does not connect to or addresses the FB directly, but rather the IDP that is bound to the FB. This way, FB A does not need to be informed if FB X is replaced by another functional block, as this can simply be achieved through re-binding the IDP with the new functional block. This useful decoupling prevents network entities to be involved in and be aware of any (autonomic) re-binding procedure that can take place during active communications. Figure 3-2 illustrates in form of a few examples how bindings based on IDPs are used.

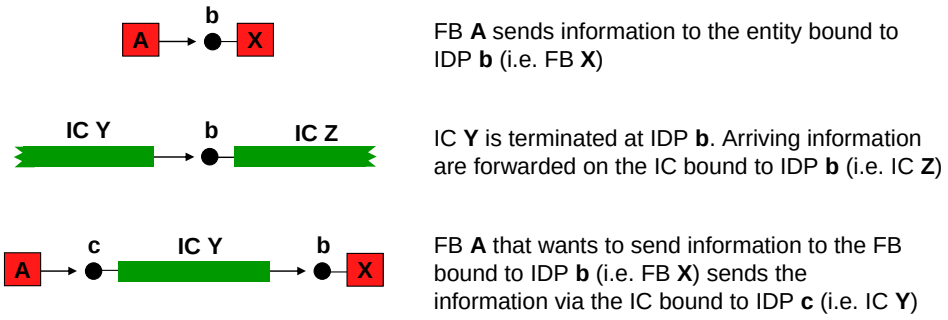



**Figure 3-2. Connecting FBs and ICs through Information Dispatch Points (IDPs)**

Since IDPs also provide functionality that is related to information processing, it could be argued that IDPs are just a special type of FBs. Nevertheless, as the ability to transparently (re-)bind ANA elements is architecturally important, IDPs are considered one of the core abstractions of ANA. In contrast to FBs, IDPs are however limited to perform only data forwarding operations (based on the bindings).

The following example (Figure 3-3) shows how FBs, IDPs and ICs interact in order to enable communication within a compartment. The FBs in this example perform some type of processing (e.g., add protocol header, perform rate control, etc.) on the data before it is transmitted to an entity in another ANA Node via an IC. The bottom IC in this example represents the service that is provided by the underlying system (e.g., the operating system providing a link and physical layer functionality). Since this IC is not provided by the network compartment considered in this example, it is shown in the imported view of the compartment. As discussed later in section 3.3.2, the IC can also represent the service provided by another, underlying compartment. In addition, the top IC shows the "communication service" created by the network compartment as it would appear to an external entity using it. Typically, the ICs exported by a compartment can be imported by other compartments. Finally, the structural view of the figure shows the internal processing elements (i.e. FBs) that execute the protocols and functions that form the "compartment stack". Note that for simplicity the figure shows a single FB but in practice there could be multiple interconnected FBs.

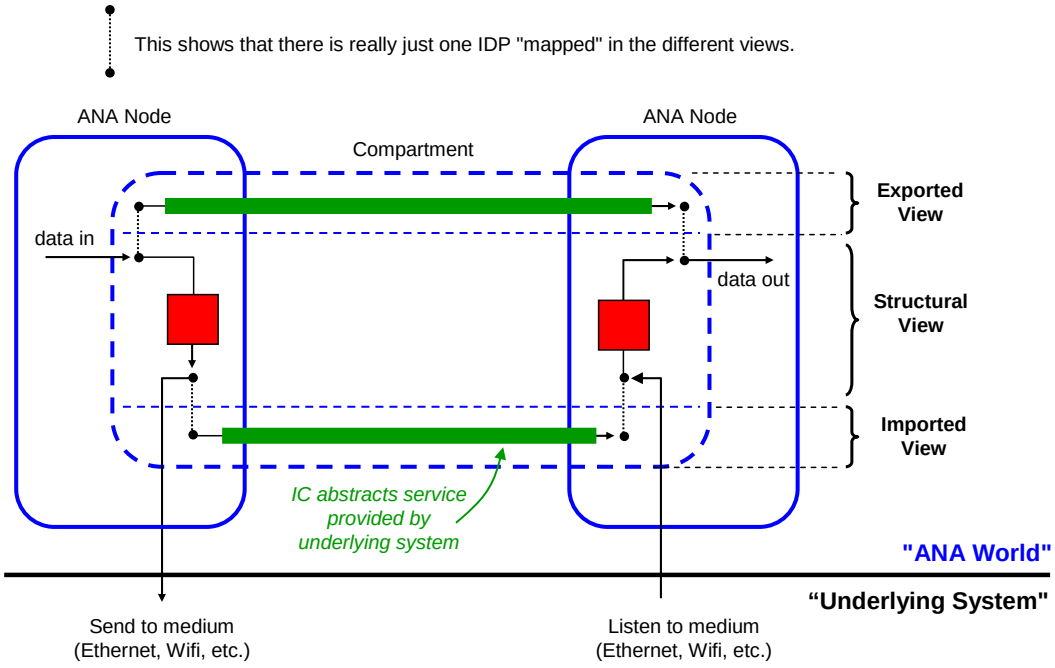



**Figure 3-3. Different Views of a Compartment: Imported View, Structural View and Exported View**

Figure 3-4 shows the conceptual or external view of how a client of a compartment sees the service that is provided by the compartment. The service provided can be abstracted by a logical IC: An external entity using a compartment can simply view the

compartment as an IDP (i.e. the entry point to the compartment) and an IC (i.e. the service that is provided by the compartment).

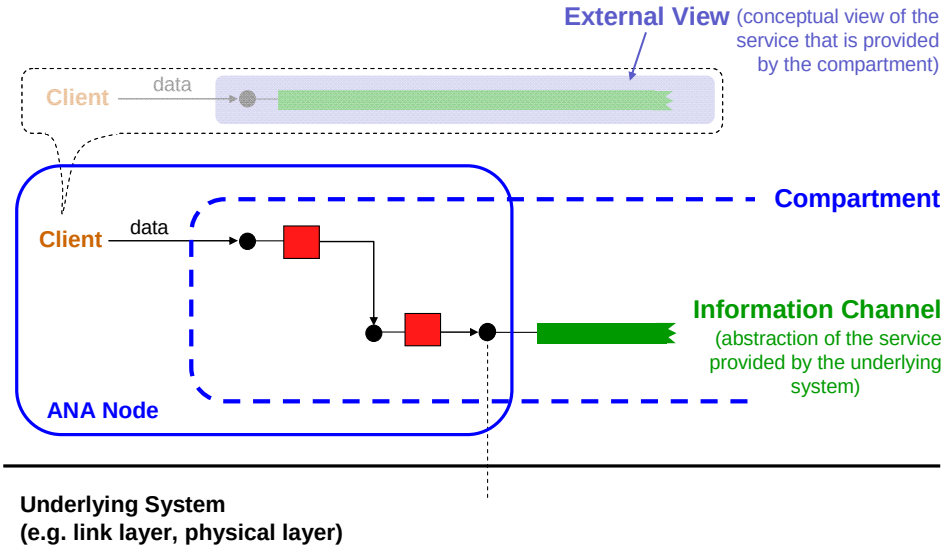



**Figure 3-4. External View of the Service a Compartment from a Client's Perspective**

As a result of the way FBs, ICs and IDPs are put together to form a compartment, compartments can also be defined by a policed set of FBs, ICs and IDPs that constitute the compartment. The policies that control the ANA elements that belong to a compartment are part of the compartment definition.

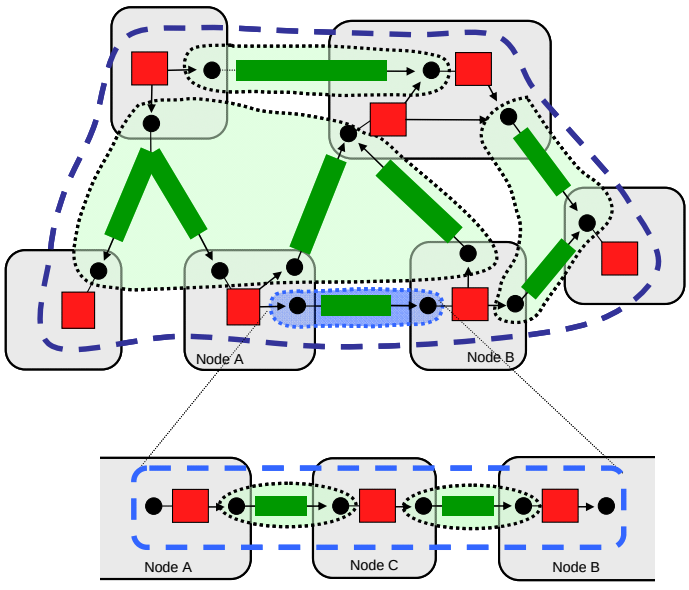



**Figure 3-5. The Compartment: A policed set of Functional Blocks (red), Information Channels (green), and Information Dispatch Points (black)**

Figure 3-5 illustrates how compartments are formed by a set of communication elements (FBs), the communication media between the communication elements (ICs), and the dynamic binding elements (IDPs). The figure also shows how an IC can actually abstract (or hide) the service provided by another underlay compartment.

### 3.1.3 Basic Compartment Functions

#### *Registration and Deregistration:*

Compartments require some type of registration or publish function that allows communication entities to become a member of the compartment. A corresponding function is needed for a member to deregister or leave a compartment. A compartment's registration and deregistration functions depend on the actual purpose of the compartment. For example, compartments that only allow certain members to join need to undergo some authentication function. Others that are completely open but provide a policy or trust domain need to verify that the (composed) FB that implements the compartment functionality conforms to the compartment. While compartment registration is in most cases expected to be explicit, there are also legacy compartments whereby the registration is implicit, e.g., when entities are statically or "by default" part of a compartment.

#### *Policy Enforcement:*

Compartments can police their members and resources - i.e. the set of communication elements (FBs), communications media (ICs) and binding elements (IDPs) that constitute the compartment. For example, a compartment can ensure, e.g., through authentication of the FBs, that only trusted FBs that conform to the compartment stack can be instantiated.

#### *Identifier Management:*

Compartments that use some kind of identifiers (i.e. names, addresses, etc.) for "naming or addressing" of individual or groups of members need to manage the identifier space. Compartments will typically associate a (pseudo) unique, compartment-local identifier to individual or groups of communication elements (FBs) during the registration phase. What kind of identifiers are used and how the compartment manages those identifiers depends entirely on the type of compartment. For example, a link-layer compartment could use MAC addresses as identifiers of individual members, whereas a compartment that encompasses a legacy IPv4 network could simple re-use the IPv4 addresses allocated to the node.

#### *Identifier Resolution:*

For compartments that use identifiers for "naming or addressing" of individual or groups of members, the compartment will also provide the necessary resolution functions for members to lookup an identifier and obtain the necessary information how to

communicate with or address the respective member(s). The result of the resolution process depends again on the type of compartment and how communication within the compartment is handled. The resolution process may simply result in another identifier (e.g., address) that is required for communication at a lower layer, or return any other hints (context) that are required to send or forward information towards the destination within the compartment (e.g., the next IC to be used or the next hop address).

Figure 3-6 illustrates an example of the internal resolution process: a compartment member requests the resolution of a peer member "A". The compartment's resolution mechanism provides the necessary information to establish a communication with the peer member "A". It returns for example an IDP labelled 'a' that stands for a communication channel towards member A. How the resolution is performed is specific to each compartment. Note that the resolution process may even be implicit, for example as it is currently the case in the Internet, where an IP address "defines" a member that is reachable, in principle.

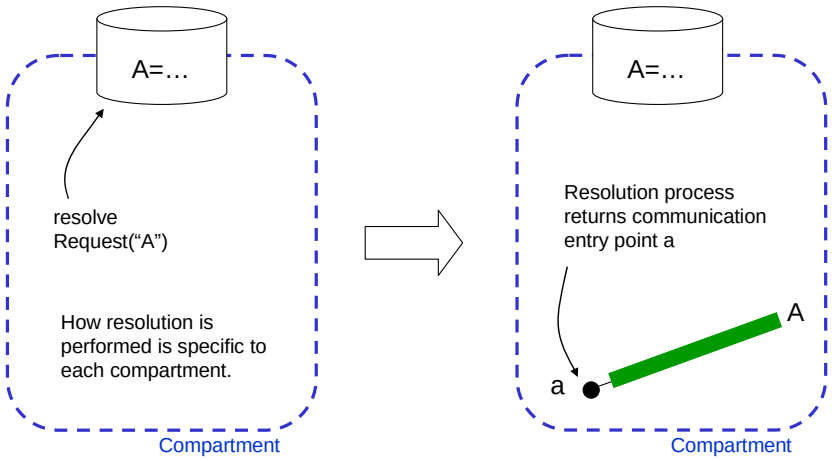



**Figure 3-6. Example of an internal resolution of a compartment member**

### *Routing:*

Compartments typically also include a routing function that is responsible to select the communication paths between any information source and destinations within the compartment. The outcome of the path selection process (i.e. the routing information) is used by the identifier resolution function to derive the necessary forwarding information upon a resolution request. Further details on the ANA routing framework are provided in Deliverable D2.1 [5].

The remainder of this section describes two particular types of ANA compartment, namely the Node Compartment in section 3.2 and the Network Compartment in section 3.3, in further detail.

## 3.2 Node Compartment

The Node compartment is present in every ANA node: it is the primary hub via which all interactions with ANA elements and compartments are initiated. It interacts with the core ANA software and has access to network interface cards (NICs) either via the core ANA software or via the operating system running on the physical device. "Communication" with the node compartment is similar to communications with any compartment, i.e. it relies on the same basic mechanisms (control protocol and APIs which are later described in 3.2.2 and 3.3.3 respectively). In a sense, the node compartment is like a "microscopic" network compartment involving just one physical device. Like any network compartment, the node compartment has explicit members and it is governed by policies (e.g., access and control rights).

### *3.2.1 Private views inside the node compartment*

For each "client" of the ANA architecture, i.e. protocols and applications using ANA, the node compartment offers a dedicated communication workspace and a private view (representation) of the ANA node. Each view may be governed by its own policies, access rules, and may have strict or loose boundaries with other views. For example, a view may include shared elements provided by other applications or protocols.

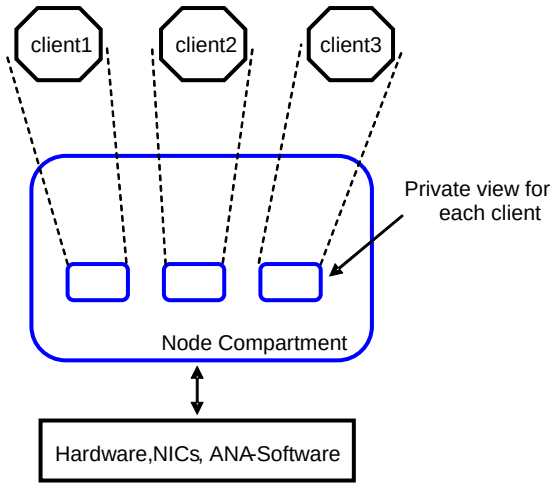



**Figure 3-7. Private views of the node compartment.**

The particularity of node compartments is that any functional block (FB) or information dispatch point (IDP) belongs to one and only one node compartment, although IDPs can be "mapped" into one or multiple network compartments. In contrast, an information channel (IC) may start and end in multiple node compartments, but a given information channel always "traverses" one and only one network compartment. Note that this implies that a node compartment has full control over the FBs and IDPs it hosts, but it does not have full control over ICs (which are a kind of "shared object").

### 3.2.2 The node compartment and control channel

The communication with a node compartment is done via a control channel and a control protocol. The control channel is platform/implementation specific and may be realized via e.g., UNIX pipes or sockets. To mask implementation details, a set of APIs provide a generic interface to connect to the node compartment. Communication with the node compartment is achieved via a control protocol which defines a set of APIs to access not only the node compartment but also any network compartment. The control protocol is used by an ANA client to e.g., access its private view, discover available network compartments, setup communication with compartments members and instantiate FBs to perform dedicated packet processing tasks. Note that section 5.4 of this document gives more details about both the control channel and protocol.

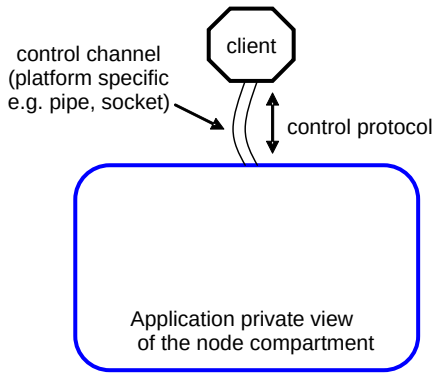



**Figure 3-8. A client and its node compartment (private view of).**

In the remainder of this section, we will assume that an application (client) can send simple requests to compartments, although the syntax may slightly differ from the detailed (and more exact) description provided in section 5.4. Note that the outcome of a request (e.g., of a function call) is typically the instantiation/deletion of IDPs and FBs in the node compartment handling the request, and may trigger the creation/deletion of ICs across network compartments.

Communication with a network compartment is typically handled via a dynamically instantiated *access object* which will be running inside the node compartment or in a dedicated framework of the ANA node (i.e. the "playground") or in the core ANA software itself. In the following example, we assume that there exist such access objects although we do not provide further insight on how and where these objects are exactly implemented. The current focus of this section is indeed to provide a basic understanding of the operation of the node compartment, and illustrate by means of an example how a basic communication setup is performed.

### 3.2.3 Example of Communication setup

Upon start-up, each client of ANA has access to a private view of the node compartment as shown on the following figure. Communication with the node compartment is done via an access object to the node compartment: it handles all (control) requests sent by the client to the node compartment abstraction. In Figure 3-9, the access objects of the node compartments can be reached via IDP *p* for client1 and IDP *v* for client2.

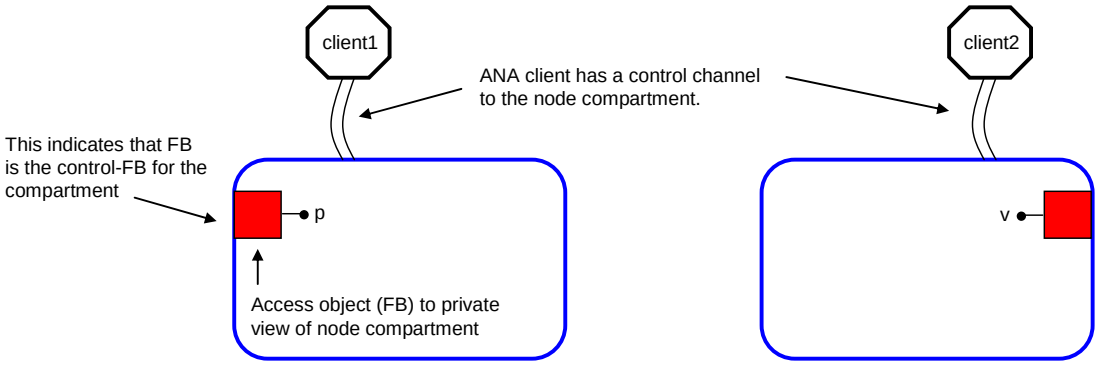



**Figure 3-9. Access object (FB) to the node compartment.**

Each ANA client has now access (via the node compartment access object) to the membership database of the node compartment. In particular, this database stores the information of available network compartments that clients can request access to. This database also stores information about functionalities potentially advertised and provided by other clients, and information about functional blocks (other than access objects to network compartments) that are provided by the ANA software, e.g., for packet encryption, data compression, transmission rate control, etc.

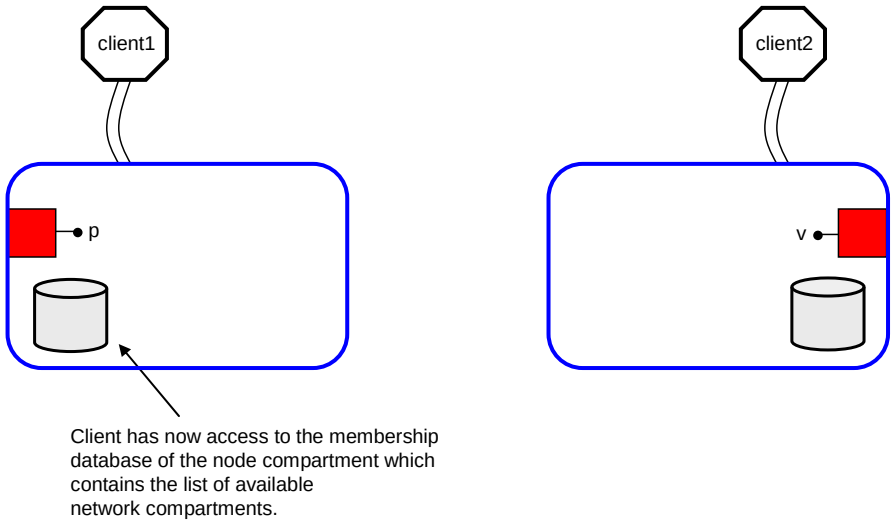



**Figure 3-10. The internal database of the node compartment.**

Assuming that the two ANA clients want to establish communication via the local Ethernet link, both ANA clients now request access (via the node compartment's access objects) to the access object of the Ethernet compartment. These requests return IDP *e* for client1 and IDP *f* for client2 respectively. At that point, both clients can communicate with the Ethernet compartment.

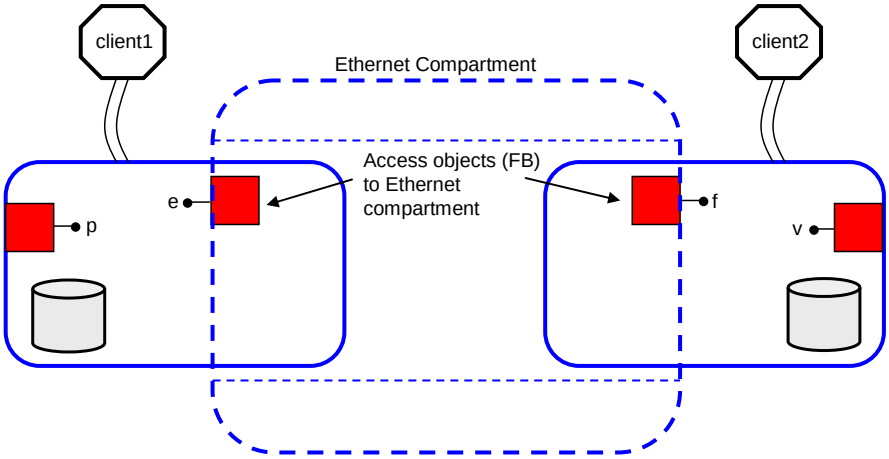



**Figure 3-11. The access objects to the Ethernet compartment.**

Note that the access object is the main interface to all the FBs that compose the "stack" for a network compartment, and is hence also referred to as the compartment's Access FB (AFB). The network compartment's AFB implements (ethernet compartment in this case) the compartment API (see section 3.3.3 for further details), which allows network compartment clients to register/publish their services or resolve services via this compartment.

The following figure show the compartment "stack" and the AFB as a separate FBs, but in practice the compartment stack could be handled by several FBs (e.g. a routing FB, a naming FB and an addressing FB).

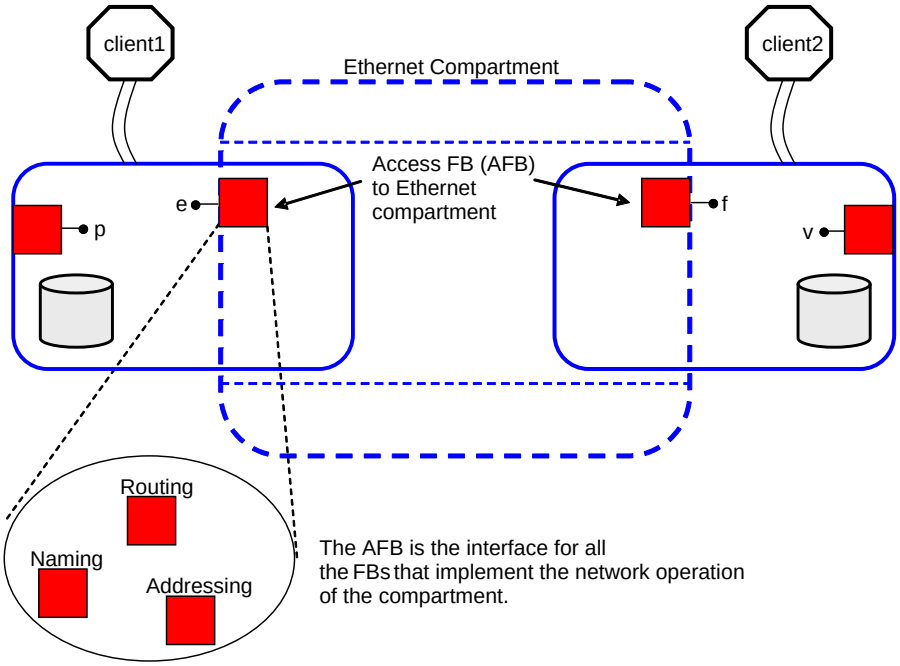



**Figure 3-12. Stack abstraction.**

When ANA client2 requests via its Ethernet compartment's AFB (bound to IDP *f*) to become reachable in the Ethernet compartment, it registers/publishes itself in the Ethernet compartment with some identifier "B". A functional block (stack or part of it) that will handle incoming data is instantiated and an output IDP *b* is created. Note that there is only one IDP *b*: it appears in the node compartment and in the structural view of the Ethernet compartment, and is "mapped" in the export view of the Ethernet compartment.

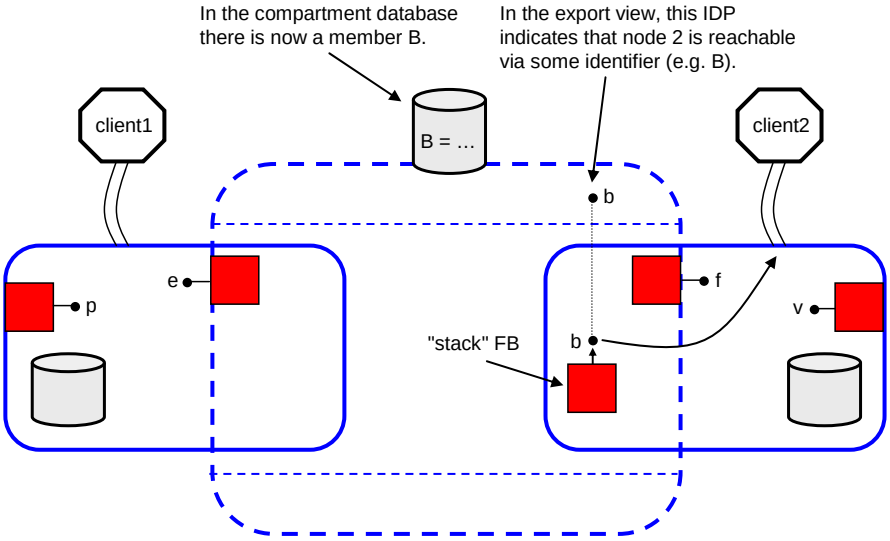



**Figure 3-13. Explicit reachability inside Ethernet compartment.**

When ANA client1 wants to communicate with the Ethernet compartment member that is identified as "B", it requests via its Ethernet compartment's AFB (bound to IDP *e*) to resolve the identifier "B". Assuming the Ethernet compartment successfully performs this resolution, an IDP *a* is created and this value is returned to the client. Data can now be sent by client1 via the IDP *a*. Note that in this example, IDP *a* is bound to another "stack" FB which will handle incoming data and perform appropriate processing to send this data to the destination.

At this point, the export view contains an IC, which abstracts the internal details of the communication: for a user of IDP *a*, the service provided by the compartment is an information channel to member B.

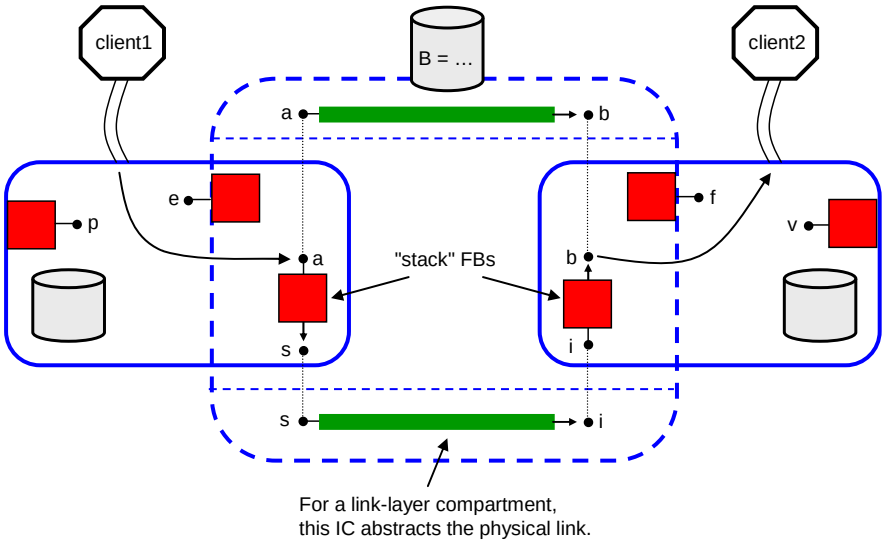



**Figure 3-14. Member resolution and path setup.**

It is important to note that in this example, the two "stack FBs" are actually connected by the physical medium (e.g., an Ethernet switch and twisted-pair cables). This low-level connectivity is shown in the import view of the Ethernet compartment by the IC linking IDPs *s* and *i*. In practice, the low-level functionality (send and receive to/from medium) bound to these two IDPs is provided by the abstraction layer of the ANA node as described in section 5 of this document. Observe that the two ICs shown in this figure are different: the IC {*s*;*i*} abstracts a physical link with no ANA functionality while IC {*a*;*b*} abstracts an active ANA communication with some processing performed by the two "stack FBs".

Finally, the picture below shows the export view of this communication in the Ethernet compartment. This shows that for client1 there is really only an information channel to client2: the internal details of what this IC really is are "hidden" by IDP *a*.

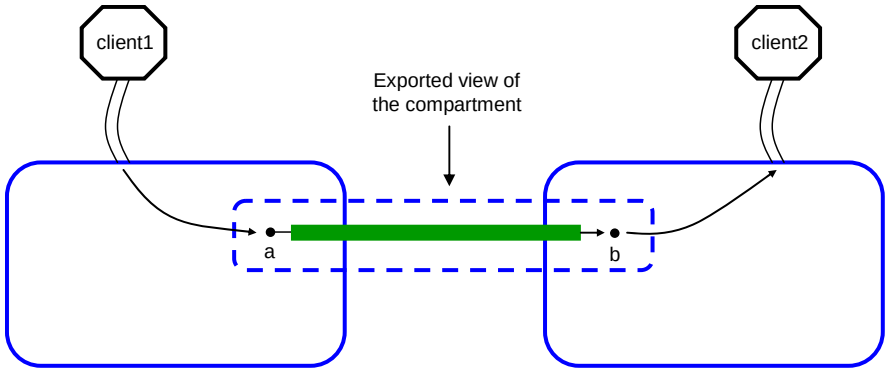



**Figure 3-15. Export view of Ethernet compartment.**

## 3.3 Network Compartments

A Network Compartment is a compartment that encompasses several ANA nodes and involves communication across an underlying network infrastructure.

Like for any compartment, network compartments also consist of a policed set of FBs, ICs, and IDPs. In case of a network compartment, the FBs providing the compartment's communication stack are typically located on different ANA nodes. Note that ANA does not distinguish between end systems and intermediate nodes – a network compartment includes both types of nodes. ICs provide the communication channel between the networked ANA nodes hosting the FBs. For network compartments, ICs typically represent the underlying communication channels (e.g., an Ethernet link or a wireless channel) over which communication between the ANA nodes takes place, or the abstractions of the service that is provided by an underlying compartment.

Examples of a network compartment are:

- A corporate network – whereby the hosts (i.e. PC and servers) and network nodes (i.e. routers) owned by the corporation host the FBs that provide the communication stack that is used within the corporate network compartment, and the network links (e.g., layer 2 links or VPNs) represent the ICs; the corporate network compartment also defines its own (private) addressing domain and defines the policy/trust domain.
- A wireless sensor network - whereby the sensor nodes host the FBs that provide the communication stack used within the sensor network compartment, and the wireless channels between the sensors represent the ICs.
- An overlay network – whereby the overlay nodes host the FB that provide the communication stack for the overlay, and the virtual links established between the overlay nodes represent the ICs.

### 3.3.1 Compartments and Layers

Network compartments can operate at various layers of the OSI model. That is, communication among compartment members can take place, for example, at the link layer, the network layer and/or the transport layer. A compartment can also combine the functionality of several layers (e.g., link layer and network layer) in one compartment.

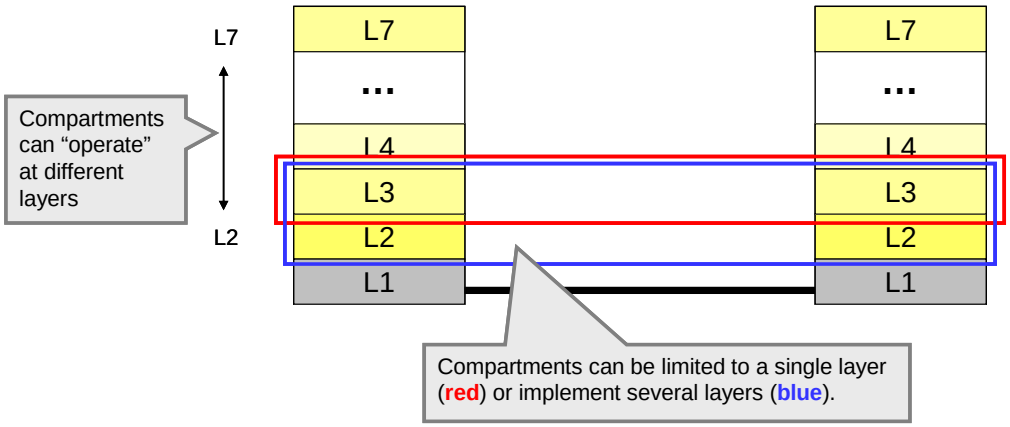



**Figure 3-16. Relation between Compartments and OSI Layers**

Compartments that operate at different layers or different scopes (e.g., local vs. global), can be "layered" on top of one another. For example, a compartment that offers end-to-end communication services across heterogeneous network domains can be "overlaid" on top of the compartments providing the functionality in each of the underlying network domains.

Unlike layers in the OSI model which rely on strict layering of layer *X* on top of layer *X*-1, compartments can be combined in much more flexible ways. As illustrated above, compartments can include semantics and functions of more than one layer (indicated by the blue rectangle), or it can provide only the functionality of a single layer (indicated by the red rectangle) or even just some aspects of a conventional OSI layer (e.g., just a control plane or management plane function). As such, compartments have much more "fluid" semantics (depending on the mission and the number of planes they span) and are less opaque than layers are in the OSI model. It is even possible that a single compartment implements a whole communication stack.

Another difference between layers and compartment is that compartments do not necessarily operate end-to-end, while OSI layers above layer 2 are all end-to-end. In other words, compartments can not only interface with other compartments in a vertical manner, but also in a horizontal way in order to provide end-to-end communication. For example, a compartment can be composed of different types of network layers (e.g., an IPv4 and an IPv6 compartment).

Finally, compartments also extend the OSI layer model by supporting administrative policies, which enable flexible and fine-grained control of the operation of compartments

and the entities that can participate, whereas layers implicitly involve all the entities that implement the (static) functionality provided by the layer.

### 3.3.2 Layering Compartments

The ability of layering compartments on top of one another depends on functionality (i.e. semantics) the upper compartment expects from an "underlay" compartment and the lower compartment can offer to an "overlay" compartment.

In order to support layering of compartment X over compartment Y, one requires either that compartment X supports the particular compartment interface of compartment Y, which then simply "encapsulates" the data and opaquely passes them through the underlay compartment, or that an appropriate "interworking function" exists between compartment X and Y (e.g., an appropriate FB) that can translate between different protocols and/or addressing schemes, or proxy communications of compartment members in the other compartment.

Figure 3-17 illustrates the case where the overlay compartment (Compartment N) directly supports the syntax and semantics of the compartment interface exposed by the underlay compartment (Compartment L1). This implies that the output interface of the "source FB" on the overlay compartment (left top corner) is compatible with the input interface expected by the FB of the underlay compartment (left bottom corner) and vice versa for the receiving in the intermediate node.

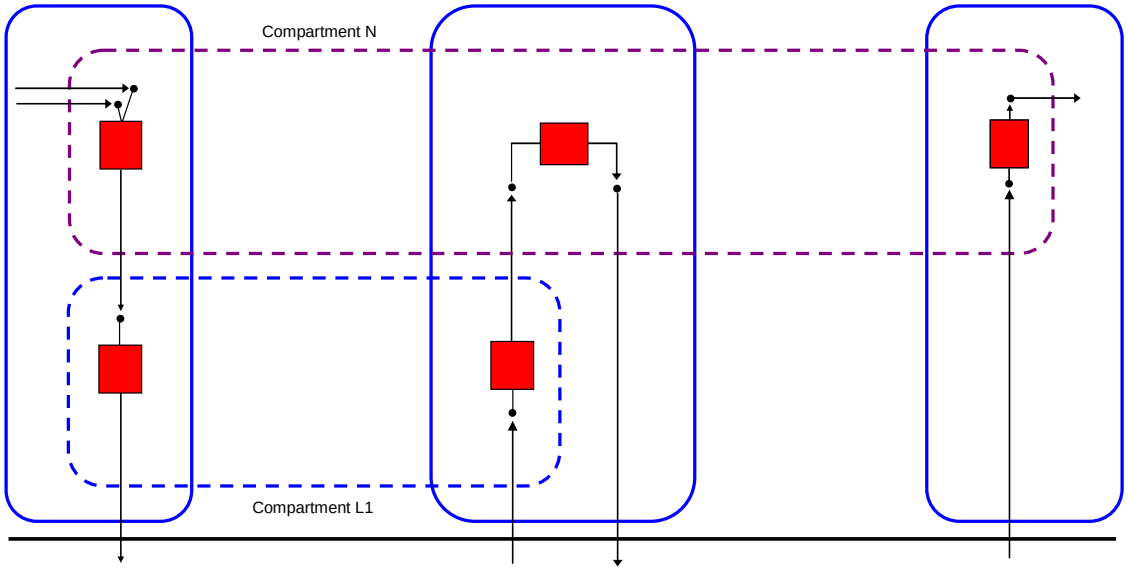



**Figure 3-17. Layering compartments in case the overlay compartment support the interface of the underlay compartment.**

Note that in Figure 3-17 it is assumed that for the communication between the intermediate node and the destination node, Compartment N is directly interfacing with the underlying system – i.e. no underlay compartment is involved.

Figure 3-18 presents the case where appropriate "interworking functions" are included in between the overlay and underlay compartment. The Interworking Functional Blocks (IFBs) are FBs that are conceptually part of both compartments. They must support the functionality required for communication in both compartments and be able to perform the necessary protocol and/or identifier translations.

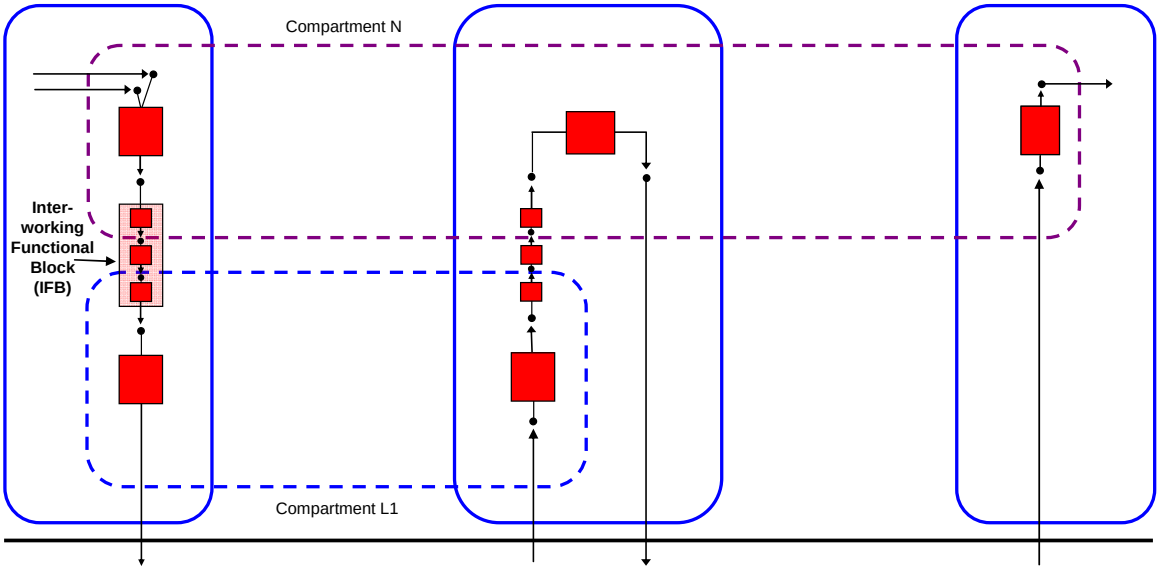



**Figure 3-18. The Interworking Functional Blocks (IFBs) enables "translate" between the syntactical and semantical differences of the overlay and underlay compartments**

The latter approach (with the interworking functions) is advantageous in case a particular compartment can serve as the overlay compartment for many heterogeneous underlay compartments. In this case, the different IFBs can be used to "translate" between the overlay compartment and the various underlay compartments. Hence in that scenario, there is no need that all compartments must support exactly the same interface. The IFBs can hide the syntactical and semantical differences of the interface supported by the overlay compartment and the interface exposed by the underlay compartment.

Further details on how inter-compartment communication is handled are provided in section 3.4.2.

### 3.3.3 The Compartment API

This section provides a deeper description of the core communication primitives supported by ANA. These primitives, referred in other documents as the *Compartment API*, constitute the fundamental and generic interface used to interact with network compartments. In particular, all compartments must support these generic primitives.

In brief, the basic objective of the compartment API is to allow un-restrained interactions between network compartments. "Un-restrained" here means that one can for example layer compartments arbitrarily since all compartments expose the same generic API. Note that the generic API does not prevent one from designing compartment-specific

primitives. However, developing compartment-specific functions reduces the "genericity property of ANA" and the relevance of having the compartment as a "generic network wrapper".

Compared to section 3.2.3 which provides a high-level introductory description of some basic communication setup scenario in ANA, the focus of this section is on the core primitives of the compartment API. Note that for simplicity, the function prototypes presented here use some kind of a *pseudo-language* while section (5.5) provides more detailed prototypes in C and/or Java.

#### 3.3.3.1 Basic Primitives

The compartment API basically offers five fundamental primitives which will be illustrated later in this document with some examples. The basic primitives are:

- **$IDP_p$ = publish ($IDP_c$, context, service)**: to request from a compartment that a certain *service* becomes reachable via the compartment (identified by $IDP_c$) in a certain *context*. When successful, publish returns an IDP ($IDP_p$) via which the service has been published.
- **int = unpublish ($IDP_c$, context, service)**: to request from a compartment (identified by $IDP_c$) that a certain *service* is no longer reachable via this compartment and *context*. The primitive returns an integer value to indicate a successful request or some error condition.
- **$IDP_r$ = resolve ($IDP_c$, context, service)**: to obtain from a compartment (identified by $IDP_c$) an information channel to communicate with a certain *service* that has been published in some *context* inside the compartment. When successful, resolve returns an IDP ($IDP_r$) via which the service can be reached.
- **info = lookup ($IDP_c$, context, service)**: to obtain further information from a compartment (identified by $IDP_c$) about a certain *service* that has been published in some *context* inside the compartment. When successful, lookup returns some information (to be detailed later) about the service being looked up.
- **int = send ($IDP_s$, data)**: to send data to a *service* which has been previously resolved into the $IDP_s$. The function returns an integer value to indicate a successful request or some error condition.

The two fundamental notions of *context* and *service* have been introduced in order to be able to explicitly specify *what* (service) and *where* (context) "inside a compartment" a certain resource must be published or resolved. In other words, the context typically provides a hint to the compartment how to resolve the requested service while the service describes a resource inside the context.

Note that all compartments like e.g., Ethernet, IP, DNS, etc, must support this generic API. In our current implementation, both the context and service arguments are simply character strings. As such, one can write "compartment-agnostic" code where there is no need to have any a priori knowledge about a certain compartment in order to use its services. This for example contrasts to the socket API where one must explicitly know

which sockaddr data structure should be used with a given "compartment": i.e., sockaddr_in for AF_INET, sockaddr_ll for AF_PACKET, etc.

Note also that this is most likely not yet an exhaustive list of primitives needed. The ongoing development activities for various network compartments types may demand further primitive to be included.

### 3.3.3.2 An Example

To illustrate these primitives and the notions of context and service, we describe a basic example in which two "IP-stacks" communicate over an Ethernet compartment (segment). Figure 3-19 illustrates the publish primitive inside a node M where the functional block IP-FB publishes itself in some Ethernet compartment by calling publish(y, "*", "1.2.3.4");

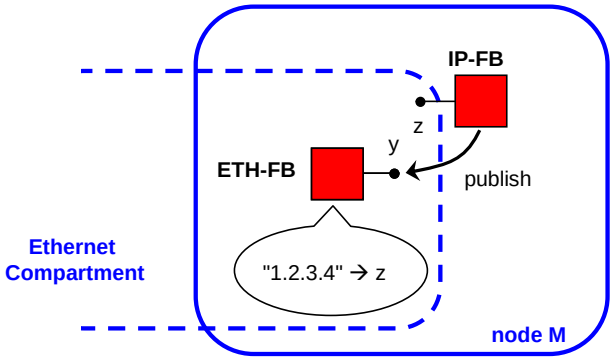



**Figure 3-19. Publish of the IP service in the underlying Ethernet Compartment**

The IDP 'y' identifies the compartment: it is actually bound to the Ethernet functional block (FB) inside node M. The context argument "*" is a well known context which specifies the largest possible context as interpreted by the compartment. For Ethernet, this typically means the entire segment. The service "1.2.3.4" is simply the IP address of the IP FB via which it wants to become reachable via the Ethernet compartment. Shall the publish request be successful, the Ethernet FB keeps a mapping between the service "1.2.3.4" and the node-local IDP z (returned by the call to publish).

Note that this publish example is somehow similar to what happens in today's computers when one configures an Ethernet interface with an IP address via the 'ifconfig' command, and this makes the IP address resolvable via ARP.

Figure 3-20 now illustrates how an IP stack inside node N can instantiate a communication channel to the IP stack in node M by calling resolve(e, "*", "1.2.3.4");

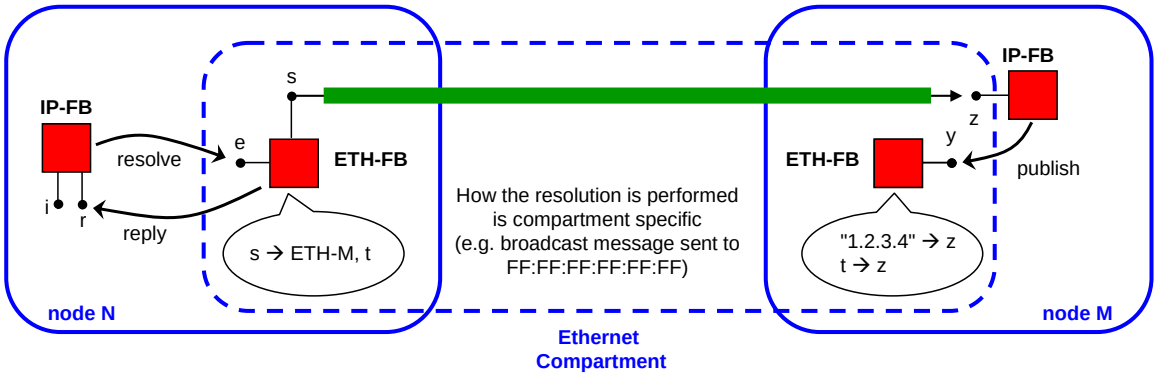



**Figure 3-20. Resolve of a peer node's IP service through the underlying Ethernet Compartment**

The IDP 'e' identifies the Ethernet FB inside node N, the context is set to "*" to specify the largest possible context (here the entire segment), and the service is set to "1.2.3.4". The successful request returns the IDP 's' which can be used by the IP FB in node N to send data to the IP FB in node M.

In ANA, how a resolve request is handled is left to the compartment. For Ethernet, this typically results in the sending of a dedicated message containing the service being searched. Note that the context "*" translates inside the Ethernet compartment into the Ethernet broadcast address "FF:FF:FF:FF:FF:FF".

In this example, when the Ethernet FB in node M receives the resolve message, it finds out that the service "1.2.3.4" has been published locally and hence replies with a dedicated message containing the *token* 't' which is used to demultiplex incoming packets to IDPs being published inside the Ethernet FB in node M. As a result, the Ethernet FB inside node N creates a local entry mapping the IDP 's' into the destination MAC address ETH-M and the token 't'. When data is sent to the IDP 's' by simply calling send(s, data), the Ethernet FB in node N adds to the data an Ethernet header with destination address ETH-M and the next header 16-bit field set to 't'.

Compared to today's network protocols, this resolution procedure is the ANA equivalent of a traditional ARP resolution where an IP address is resolved into an Ethernet address.

### 3.3.3.3 The Context and Service Arguments

In this section, we briefly detail how the *context* and *service* arguments are handled by network compartments. First, it is important to note that the context is interpreted by the compartment while the service is not interpreted. For example in the previous example, the context "*" is interpreted by the Ethernet compartment to be "FF:FF:FF:FF:FF:FF" while the service remains an opaque argument. For an IP compartment, the "*" context would typically be interpreted as being the broadcast address "255.255.255.255".

Note that the context can also be fully specified when calling the primitives of the compartment API. For example in the previous scenario, the IP FB in node N could specify the MAC address in which the service "1.2.3.4" should be resolved. If say the MAC address of node M is 00:11:22:33:44:55, one could call resolve(e,

"00:11:22:33:44:55", "1.2.3.4") to restrict the resolution request to the Ethernet FB in node M. While this makes little sense with the Ethernet compartment, one would typically set the context while resolving some target transport service via the IP compartment by e.g., calling the primitive resolve(i, "1.2.3.4", "tcp").

Note that in addition to the "*" context, we also introduce the well-known context "." to specify that the scope of a publication, resolution, or lookup request must be restricted to the FB handling the request. This permits to setup node-local communications with, for example, the context "." being interpreted as the loopback address "127.0.0.1" by an IP FB.

The handling of the *service* argument is currently simpler. That is, the resolution of a service currently follows an exact match strategy where the service being searched must exactly match the service being published. This is currently sufficient for Internet-like communications where e.g., a service published in the Ethernet compartment is an IP address like "1.2.3.4" and a service published in the IP compartment is typically a transport protocol description such as "udp" or "tcp". Note that the ANA API imposes no restriction on how the service is actually matched so that, if needed in the future, one can potentially extend the matching strategies with more advanced mechanisms.

#### 3.3.3.4 The Lookup Primitive

So far, we have restricted our examples to the resolve primitive which, when successful, instantiates a communication channel to the destination service. However, in some cases one may not want to obtain a communication channel or a compartment may simply not provide communication channels. This is for example the case with the DNS which resolves DNS names into IP addresses but does not create a communication channel to a name being resolved.

To support such scenarios, we introduce the lookup primitive which has the same prototype as the resolve primitive except for the return value. While the resolve primitive returns an IDP to a communication channel, the lookup primitive "simply" returns some *information*. This information may contain multiple TLV (Type-Length-Value) items such as contexts, services, and IDPs, which can potentially be used in follow-up resolve and/or lookup requests. For example, a DNS lookup request for a certain name may return a new context such as "1.2.3.4" or "2001::1" which could be used in a further resolution request via respectively the IPv4 and IPv6 compartments.

How exactly this information should be handled is out of scope of the compartment API operation and is left to the entity performing the lookup request and getting back the information. However, as a next step we plan to provide more detailed guidelines describing how the lookup information should be handled. Our objective is that these guidelines can be used by developers in order to implement automatic procedures for handling the information returned by a lookup request.

## 3.4 Communication Paradigms

### 3.4.1 Intra-Compartment Communication

One common characteristic of ANA compartments is that all members agree on some common set of operational and policy rules. This "recipe" includes the communication principles, protocol(s) and policies to be used for intra-compartment communication. It defines for example:

- How communication elements are identified (i.e. naming and addressing).
- How identifiers are resolved (i.e. name lookup or address resolution).
- How to route information inside the compartment (i.e. routing).
- How to send/forward information (i.e. forwarding).
- What protocol to be used to communicate between peer elements.

Figure 3-21 illustrates a scenario where a set of ANA nodes (indicated in yellow) form a network compartment in order to communicate with each other. All nodes that want to be part of the compartment must provide the compartment stack (indicated by the blue boxes) - i.e. every node has to support the necessary functionality (i.e. functional block) to join the compartment and communicate according to the compartment's communication scheme. Nodes that provide the required functionality are able to operate according to the compartment rules, and hence can participate in the compartment (indicated by the blue circle).

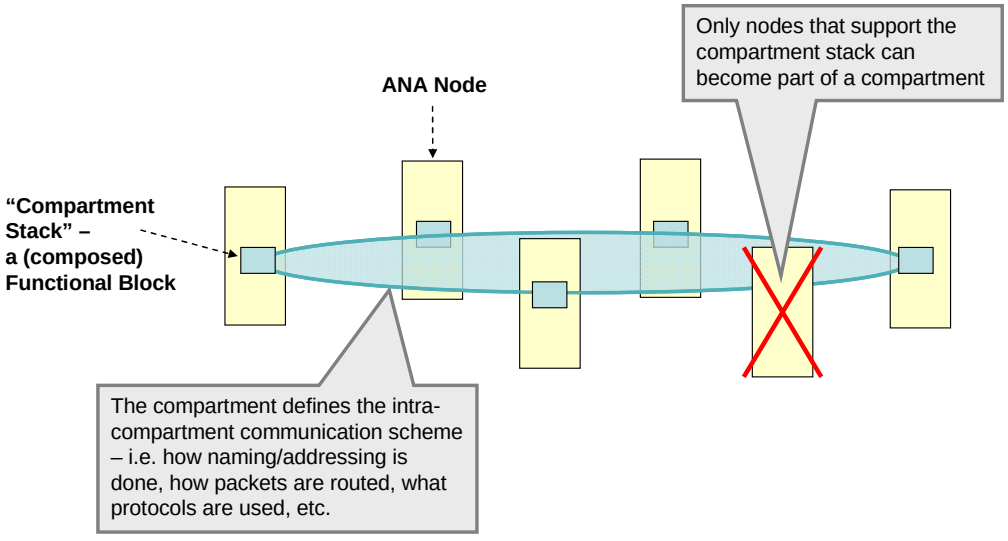



**Figure 3-21. Intra-Compartment Communication**

How communication inside the compartment is handled is entirely up to the compartment. Depending on the type and purpose of the compartment, different

communication schemes can be applied. For example, a compartment that intends to provide the communication context for a Wireless Sensor Network, where the main goal is to aggregate sensor information on low-power devices, can implement a completely different communication scheme (i.e. addressing, routing, etc.) and protocol than a compartment that targets end-to-end communication among user terminals, such as for voice communication. While the former can use a very simple addressing and routing scheme that is able to aggregate the sensor information, the latter requires a naming or addressing scheme that supports end-to-end communication and a routing scheme that minimises end-to-end delay and jitter.

Since compartments have full autonomy on how to handle intra-compartment communication, there are no global invariants that have to be implemented by all compartments or all communication elements.

#### 3.4.1.1 Compartment Multi-homing

ANA nodes can participate in multiple compartments. A node simply needs to support the necessary compartment functionality (i.e. the compartment stack) and join each of the compartments.

Figure 3-22 illustrates an ANA node that is multi-homed (i.e. part of several compartments at the same time). This enables the node to communicate with members of both compartments.

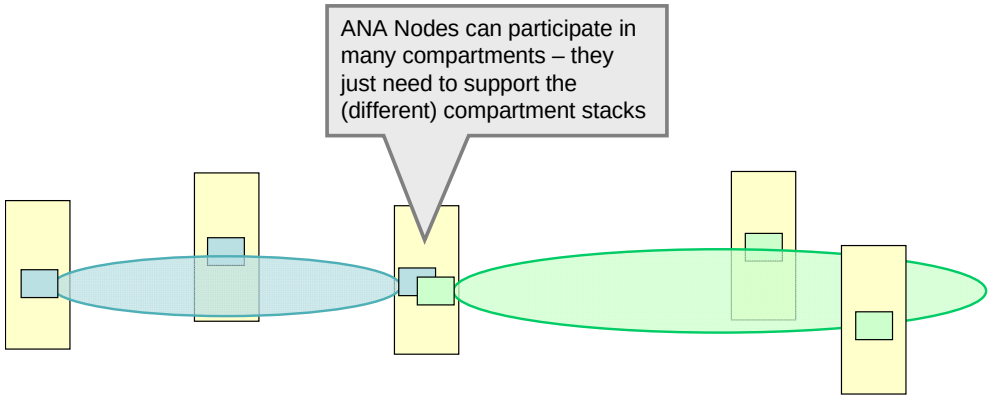



**Figure 3-22. Multi-homing in several Network Compartments**

It should be noted however that the fact that one node is multi-homed does not necessarily permit any of the other nodes to communicate through this node with ANA nodes in other compartments. In order to enable inter-compartment communication between ANA nodes of different compartments, special interworking functionality must be provided that allows translation or proxying of communications between the different compartments.

Further details on how inter-compartment communication is handled within ANA are provided in the following section.

## 3.4.2 Inter-Compartment Communication

The issue of inter-compartment communication in ANA is discussed in this section. An analysis of various inter-compartment communication scenarios is followed by a set of principles or recommendations on how to best handle inter-compartment communication in ANA.

### 3.4.2.1 Analysis of Inter-Compartment Communication Scenarios

This section analyses different inter-compartment communication scenarios, whereby communication entities of adjacent compartments want to communicate with each other. The issues, difficulties and limitations of direct communication between peers across compartment boundaries are highlighted.

Note that in this section an identifier is simply considered some sort of a "attribute" that identifies a communication element (e.g., a FB, a network node, etc.) with regard to some semantics. Example semantics involve:

- host/interface location (e.g., an address as in IP)
- host entity (e.g., a node/host id as in HIP)
- host role (e.g., a service identifier as in port numbers)
- host property (e.g., OS platform, performance capability)
- combined identifiers (e.g., network location and host role)

#### *Case 1: Without a common identifier space or overlay compartment*

The following figure shows a scenario whereby two communication elements of adjacent compartments want to communicate with each other. In this case, it is assumed that the two compartments operate completely independently (for example, one compartment based on IPv4 and the other based on IPv6).

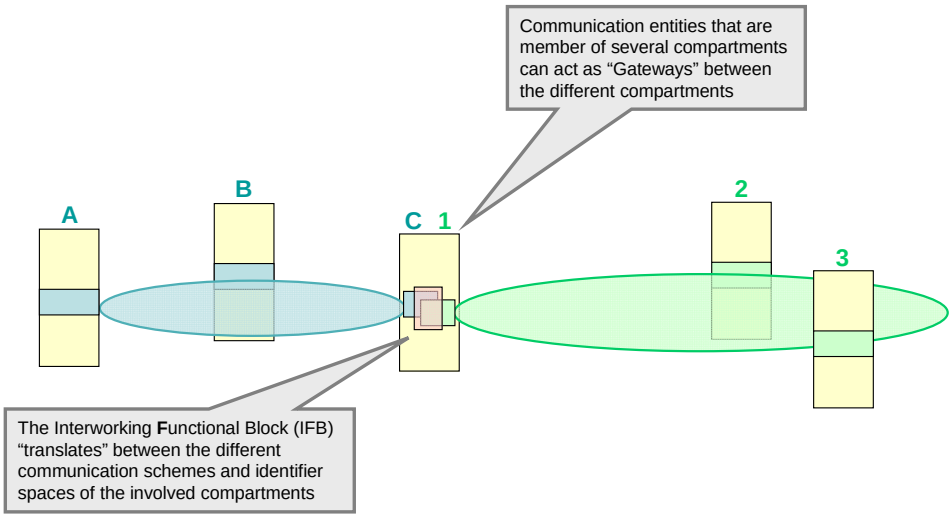



**Figure 3-23. Inter-compartment communication scenario without a common identifier space or overlay compartment**

Since the two compartments are completely independent, it must be assumed that there is no well-known mapping between the different communication syntax and semantics known to the communication elements of the different compartments. Only the gateway entity, which is part of both compartments, is able to communicate in both compartments.

This case further assumes that there exists no other, higher-level common identifier space (e.g., DNS) or common overlay compartment (e.g., a common transport protocol) that offers the peer entities of the different compartments a common syntax and semantics for the communication.

The following examples illustrate through real-life networking scenarios cases where such type of direct communication between peers of adjacent compartments occur.

***Example 1:***

For example 1, the left (blue) and the right (green) network compartments illustrated in Figure 3-23 are envisioned to be of the same type (i.e. both use the same protocol suites) and they only use different identifier spaces (e.g., addresses or ports). Then, the gateway entity "**C/1**" only needs to do identifier translation. The required translation functionality is limited to the protocol layer within which the identifiers need to be translated. In the most typical case that will be the network layer, which makes the gateway entity "**C/1**" a NAT box, but in theory, this translation could also occur at any other layer (e.g., the link layer) or a combination of layers (e.g., network plus transport layer).

***Example 2:***

For example 2, the left (blue) and the right (green) network compartments illustrated in Figure 3-23 are envisioned to be of similar type, but not the same. For example, both compartments have similar protocol suites in terms of numbers of layers and/or functionality, or provide a similar "service" to the clients. Still, translation from one protocol suite to the other (e.g., TCP/IPv4/TokenRing and TCP/IPv6/Ethernet) is doable.

However, in this case the gateway entity "**C/1**" must provide more complex translation functionality than simple identifier translation. It must also be able to perform protocol translations at one or more layers. A typical example of this is a gateway that can translate IPv4 to IPv6 headers and vice versa. Note that although this function is not very difficult to implement, some loss of functionality may be encountered during the translation (e.g., IPv6 extension headers might need to be dropped if similar options do not exist in IPv4). Therefore translation is limited to support the common denominator of the features of the two protocols.

***Example 3:***

For example 3, the left (blue) and the right (green) network compartments illustrated in Figure 3-23 are envisioned to be of completely different nature. For example, the communication stacks are completely different both in terms of number of layers (one compartment supports Layer 2 and Layer 3, whereas the other supports only L2 and L4) and in terms of protocols that are supported by each layer. Even in the case there are common layers between the adjacent compartment (common denominator), the compartment stacks may use very different protocols, both syntactically and semantically (e.g., one provides a best effort unicast service and the other provides an episodic publish-subscribe type of functionality), and identifier semantics (e.g., one uses identifiers that encode topological information and the other uses identifiers that encode service capability). In such a case, the communication entity "**C/1**" needs to have a very complex translation capability that not only spawns across all the common denominator of the layers of both stacks, but also needs to be able to collapse/expand layer functionality from one stack to the other.

In addition to the complexity encountered so far, there is also the great challenge of naming, address and routing that needs to be resolved. As it is assumed for this type of inter-compartment communication (Case 1) that the communication entities of the left compartment (blue) do not understand the identifier syntax and semantics of the right compartment (green), the gateway entity "**C/1**" needs to be able to make arbitrary decisions based on some external information (e.g., policies) how to propagate data from the source entity through the one compartment and then to the destination peer in the other compartment – without common addressing and routing semantics.

The following **analogy** points out the difficulty and issues of direct communication between peers of adjacent compartments (Case 1) in case there is no common identifier space or overlay compartment (e.g., DNS) available:

> *Two people that need to communicate without speaking each other's language nor having a common language through which they can communicate. To resolve the issue, they need to find some third-party (e.g., "Babel fish" as described in "Hitchhiker's Guide to the Galaxy") that will be able to understand both languages.*

***Conclusion:***

End-to-end communication across heterogeneous compartments without a common identifier space or overlay compartment is only possible if the peering compartments have a well-known translation function, both for the syntax and the semantics of the various identifier spaces and communication schemes.

### *Case 2: With a common overlay compartment for the communication peers*

The following figure shows the same scenario, but under the condition that both communication peers are members of a common overlay compartment (e.g., a common DNS compartment or a common HIP compartment). This reduces the problem of direct communication between the peers of the adjacent compartments to the problem of routing and forwarding – as the peer entities have a common communication context.

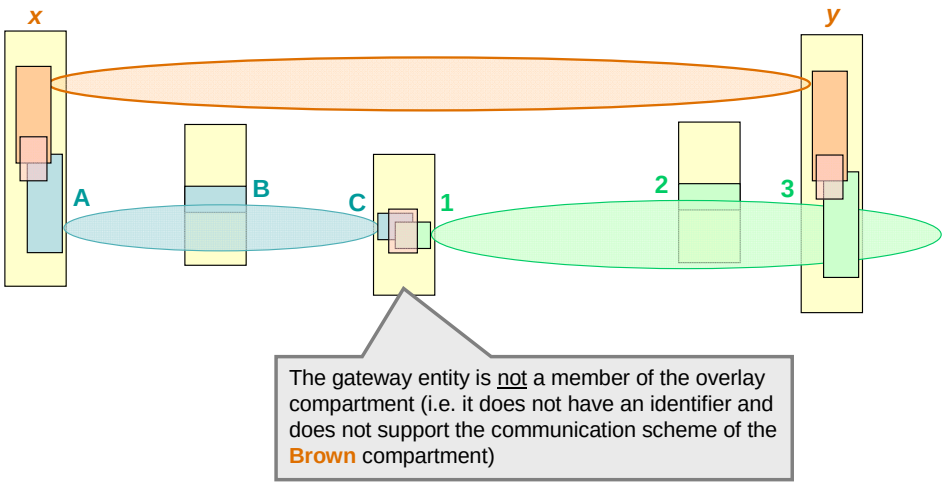



**Figure 3-24. Inter-compartment communication scenario whereby the communication peers are members of a common overlay compartment**

This case assumes that communication entity "**A**" of the left compartment (blue) and communication element "**3**" of the right compartment (green) are also members of a common overlay compartment. The common overlay compartment gives the communicating peers ("***x***" and "***y***") a common identifier space that allows them to directly address each other, and a common communication context which allows the end hosts to understand the syntax and semantic of the exchanged data.

As the gateway element "**C/1**" is not part of the overlay compartment in this scenario, it is not able to understand the syntax and semantics of the communication at that "layer". The gateway is hence is not able to take this into account when translating between the different underlay compartments. However, most importantly, the fact that the gateway is not visible at the overlay level prevents the underlay routing mechanisms to take advantage of the routing knowledge of the overlay compartment (e.g., which gateway to choose in the underlay compartment to optimise the overlay routing path).

*Conclusion:*

As the gateway element "**C/1**" is not part of the overlay compartment, the routing knowledge that is available at the overlay compartment (i.e. how to get from "***x***" to "***y***"), cannot be used to identify the gateway node as an intermediate hop between two underlay compartments. The lack of end-to-end routing knowledge at the underlay compartment level leads to inefficient routing in case of inter-compartment communication.

### *Case 3: With a common overlay compartment that includes the gateway entity*

The following figure shows the same scenario, but under the condition that the gateway entity "**z/C/1**" is also visible at the overlay compartment level. As a result, when "***x***" want to send packets to "***y***", the underlay compartment can make use of the routing knowledge at the overlay level. Since the "next hop" in the overlay compartment, i.e. the gateway entity "**z/C/1**", is also part of the left underlay compartment (blue), routing in the underlay becomes purely an intra-compartment issue. The routing knowledge of the overlay level is used by the underlay compartment to efficiently transmit the data from the source entity "**x/A**" to the correct gateway entity "**z/C/1**".

This case also includes the scenarios where the gateway entity does not support the actual communication scheme (e.g., data plane protocols) that is used in the overlay compartment, or where the overlay compartment is only used for addressing and routing purposes. For example, a gateway entity may simply "translate" between the different underlay compartments (without involving the "compartment stack" of the overlay compartment). The classical "bridge" between two distinct LAN technologies (e.g., IEEE 802.x and IEEE 802.y) using the same address space (i.e. a common overlay compartment for addressing) is an example of such a scenario. The key difference to the case 2 type scenarios is that the addressing and routing knowledge of the overlay compartment is used to "translate" between the different underlay compartments.

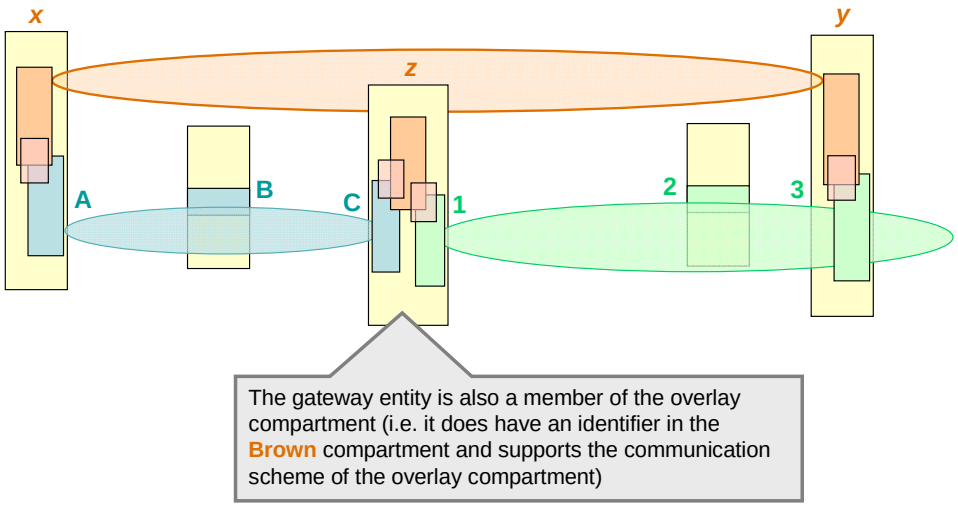



**Figure 3-25. Inter-compartment communication scenario whereby the communication peers and the gateway entity are members of a common overlay compartment**

*Conclusion:*

As the gateway entity is now part of the overlay compartment, the routing knowledge of the overlay compartment (i.e. what is the "next hop" from "***x***" to "***y***") can be used to efficiently route and forward the data within and across the underlay compartments. Hence, the availability of a common overlay compartment that not only includes the

communicating peer entities, but also the gateway entities (e.g., "**z/C/1**") enables efficient inter-compartment communication across heterogeneous underlay compartments.

#### 3.4.2.2 Principles for Inter-compartment Communication

From the above analysis of different inter-compartment communication scenarios, the following principles or recommendation can be derived:

1. *There is a need for a common compartment among communicating peers*

   Since compartments define the syntax and semantics for communication within a certain context, peers that want to communicate with each other must be part of a common compartment. Otherwise, they are not able to process (understand) the format and actual information carried in the exchanged data.

2. *It is advantageous to have a common "interworking" or overlay compartment*

   The use of a common "interworking" compartment that provides a uniform context for the communication among its members, allows hiding of potential heterogeneity at lower layers (e.g., different underlay compartments). As indicated in the conclusion above, a common overlay compartment that includes the gateway entities of the various underlay compartments also allows for efficient end-to-end routing at the underlay level.

   The fact that ANA support flexible and dynamic composition of communication stacks facilitates the introduction of such interworking "shims" as the need arises.

3. *There is a need for "interworking functionality" between adjacent compartments*

   As indicated in the analysis above, inter-compartment communication among directly adjacent or layered compartments that are of different nature require appropriate interworking functionality, which allows "translation" between the different communication syntax and semantics of the compartments. Since the required interworking functionality depends entirely on the compartments that need to interwork with each other, there is no need for a common interworking function between all compartments along the end-to-end path – i.e. different interworking functions (depending on the compartments involved) can be provided.

As illustrated in the analysis above, the focal point of the inter-compartment communication discussion are the gateway entities and the Interworking Functional Blocks (IFBs) that enable direct interworking between adjacent compartments. IFBs are conceptually part of directly peering compartments, and hence are able to "translate" between the different communication syntax and semantics of the different compartments. The IFBs are, for example, responsible for the mapping of the identifiers (i.e. name/address) as well as the translation/capsulation of the communication protocols.

Figure 3-26 shows in more details how inter-compartment communication is handled in case of a common "interworking" or overlay compartment is used. The example is based on the same scenario as the one illustrated in Figure 3-25. In this scenario, there are three entities involved. Two peer entities, namely "***x*/A**" and "***y*/3**", that want to communicate with each other despite the fact that they are not part of a common underlay compartment

(e.g., because they don't share a common link layer technology), and a gateway entity "**z/C/1**", which is part of both underlay compartments.

In order to allow for the peer entities to talk to each other, they are also part of a common overlay compartment. This offers the peer entities a common communication context, which allows the peers to address and resolve each other, and gives them a common communication protocol. Without a common communication context, the peer entities would not be able to indicate with whom they want to communicate, nor would they have a common understanding on what the exchanged messages mean.

The fact that the gateway entity is also part of the common overlay compartment facilitates end-to-end routing across the heterogeneous underlay compartments. It allows the peer entity "***x*/A**" to resolve the gateway entity "**z/C/1**" as next-hop at the overlay level. The IFB between the overlay and underlay compartment at the sender side (left) is thus able to map the 'next-hop identifier' at the overlay level ("**z**") to a meaningful 'destination identifier' in the underlay compartment ("**C**"). Note that in case the next-hop at the overlay level is not visible at the underlay level, the underlay compartment cannot resolve the destination entity in the underlay and hence can only forward/transport the data based on default routes or via some broadcast mechanism.

In addition to the identifier mapping, the IFB is also responsible to handle any necessary protocol translations or the encapsulation of the payload for transmission of the information in the underlay compartment. On the gateway entity and the receiving end, the IFB performs the opposite functions, namely it decapsulates the information or performs the necessary protocol translations between the underlay and the overlay compartment.

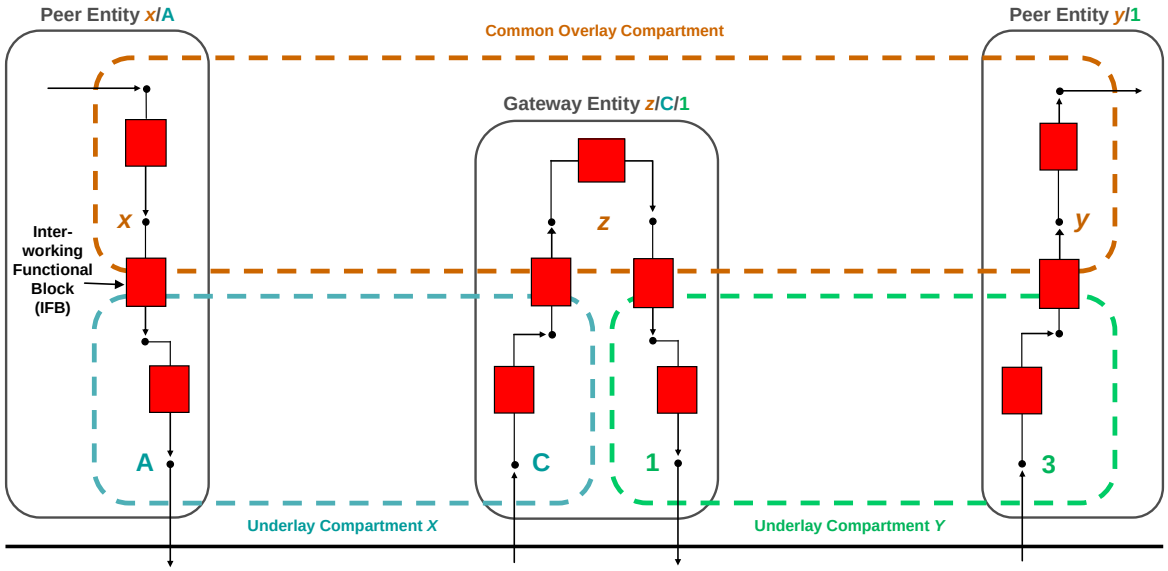



**Figure 3-26. Inter-compartment communication scenario with a common overlay compartment.**

# 4 AUTONOMICITY IN ANA

## 4.1 Information Management

Achieving self-* properties requires the exchange of information of different types. For instance, self-configuration requires that an entity has knowledge about its own configuration, that it knows about components and resources which are potentially available for interaction and usage, and that it has reasoning capabilities about the impact of configuration changes. Similarly other self-properties depend on the availability of information for the establishment of communication paths and optimization of network operation.

There are three subclasses of information of particular relevance within the ANA architecture:

- configuration information,
- signalling information, and
- monitoring information.

***Configuration information*** is used to configure a device or function. The configurations of a function or device may also change if signalling messages are received (e.g., queuing scheme if a reservation message passes a router). To distinguish the different information types, the usage of the information is relevant. For signalling the information is used to establish a communication. Configuration changes may occur as a consequence, but are not directly specified. In contrast to this, configuration information contains direct instructions to change parameter of devices or functions. Examples for configuration information are SNMP-or NETCONF-based messages.

***Signalling information*** comprises essentially control information. Signalling information is sent to devices or functions in order to establish or supervise a communication. Any kind of routing messages are examples of such information. Other examples include error, flow, and congestion control for data transportation. Messages for establishing TCP connection (SYN, FIN, ACK), resource reservation messages as used in RSVP and messages of the recently defined NSIS (Next Step in Signalling) protocol are considered as signalling information. Also SIP messages and other protocols that establish a communication on higher layers fall into this category.

***Monitoring information*** relates to information characterizing the dynamic behaviour of a system. Typically one can monitor the internal state of a network entity, e.g., the amount of free memory, current queue length, and available bandwidth. But monitoring information also includes triggering events like some observable value reaching a threshold, occurrence of packet bursts, suspicious sequence of packets indicating a potential intrusion, etc. Such information is gathered by measuring the system, either

passively (counting forwarded packets, reading some variables) or actively by generating probe traffic.

## 4.2 Network Monitoring

Autonomic networks are systems that provide self-management capabilities including self-configuration, self-optimization, self-healing, and self-protection (taking preventive measures and defending against malicious attacks). These functions are performed by feedback control loops.

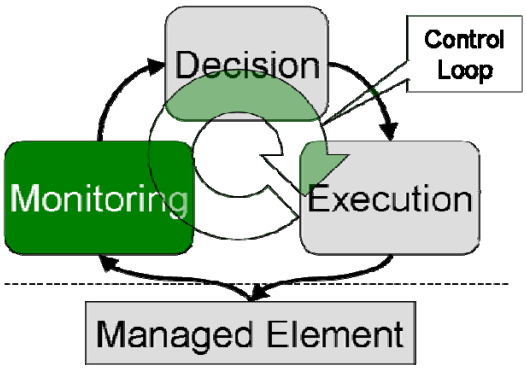



**Figure 4-1. The Network Monitoring Component**

Monitoring is the element responsible for measuring the parameters of the controlled system that are relevant for the function under control (e.g., current load for performance optimization, link availability for fault tolerance, etc.).

Current approaches to solving measurement problems often assume minimal support from nodes and protocols (e.g., active measurement) or place the entire burden on the nodes (routers) to support heavy-weight mechanisms (e.g., NetFlow or passive measurements in general). In the ANA monitoring framework, we argue that measurement must consider solutions which enlist the cooperation of nodes (routers), protocols, and tools.

In addition, monitoring has been considered a static function in the past. In the ANA project, we aim at implementing a dynamic and adaptive monitoring system. As a consequence, the decision module starts and controls the monitoring system. That is, whenever necessary, the parameters under consideration are dynamically adapted to the needs of a given module.

In the context of ANA node architectures, monitoring is implemented as a generic functional block. When other functional blocks require monitoring information for their decision processes, the monitoring functional block initiates and orchestrates the data collection and distribution of monitoring results.

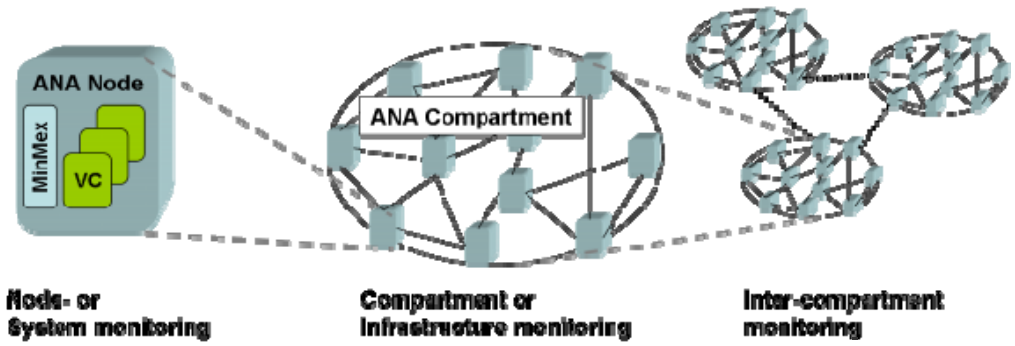



**Figure 4-2. Monitoring at various levels**

Conceptually, monitoring is always done inside a compartment; i.e., all nodes that participate in a monitoring task form a compartment. From an aggregation or information point of view however, ANA distinguishes three levels of monitoring:

- ANA node-level: Monitoring is triggered locally by the node itself and data collection, computation, and evaluation functions are invoked within the node. Node-level monitoring falls into the field of system monitoring
- ANA compartment level: Conceptually, compartment-level monitoring can be considered as infrastructure monitoring. At the compartment-level, it is orchestrated by decision makers responsible for the health and performance of a compartment. The decision makers issue for example data collection requests, or monitoring-data requests to sensing nodes or data collection points in the compartment. Data collection points also push monitoring data and event notifications to the decision makers. The monitoring includes observation of external events and event notifications to other compartments.
- Inter-Compartment level: Traffic flowing between compartments is monitored by inter-compartment adjacent nodes for the detection of compartment failures, QoS violations, security threats or other failures.

The ANA monitoring framework focuses on the architectural changes required to make the network more observable, and therewith more controllable.

## 4.2.1 Key monitoring concepts

### *Dynamic and adaptive Monitoring*

As described above, each functional block exposes parts of its internal information. The task of the monitoring functional block (or monitoring service) is then to collect and assemble the available information as well as to orchestrate the distribution of this information. External functional blocks (like for example a routing task or anomaly detection function) are then able to use this data.

The monitoring framework is specifically designed to dynamically adapt to changes in the monitoring tasks. If for example, additional information is required to detect traffic

anomalies, the data capturing is updated "on-the-fly". Also, to limit the resource consumption of the monitoring service, the service must be able to adapt autonomically (for example by adapting the capturing rate or by applying data sampling techniques).

Secondly, the monitoring functional block must be able to adapt in its perimeter. Whenever necessary, additional monitoring sensors or monitoring processing units are installed and hence added to the monitoring compartment.

### *Avoiding active measurements*

In the current Internet, monitoring tools apply active and passive measurement techniques. Passive monitoring captures system parameters (e.g., traffic volume and flow counts) and helps determine the causes of network performance problems as opposed to active measurement, which provides insights into the effects of network problems.

With the current Internet architecture, some performance parameters can only be tested by applying active measurements. Note that active monitoring implies the interaction with the system, e.g., traffic is injected into the network and captured at a destination node.

In an ANA network, we aim at a system, where active measurements are not necessary anymore. By combining passive measurement data, the decision process should be able to determine the same level of detail as obtained with active monitor probes.

### *Monitoring compartment*

In a self-managing network a.k.a. an ANA network, some nodes may need to trigger monitoring functions on one another or on systems dedicated to monitoring tasks. We call the set of distributed functional blocks that play a role in a monitoring service a monitoring compartment. The triggering nodes have to be aware of the monitoring capabilities of the nodes in the compartment and must be able to locate the monitoring point of interest on the topology.

Such a platform is an enabler for realizing a traffic monitoring compartment, which is a specialized case of a monitoring compartment. A traffic monitoring compartment is a set of functional blocks participating in a traffic monitoring objective in a distributed fashion across an ANA network. Note that a monitoring compartment might span over one single compartment or over multiple compartments. When multiple compartments are involved we speak of inter-compartment monitoring.

## 4.3 Network Management

Network-Management in an autonomic network requires the co-operation/collaboration of a number of management functions that are distributed in the network nodes. Traditionally network management is divided into five so called management planes namely: Fault-management, Configuration-management, Accounting-management, Performance-management and Security-management. This is known as the FCAPS management Framework.

In the course of this research, we seek to create a partial picture on how and where to place the following functions in an ANA node(s):

- Configuration management functions,
- Accounting management functions,
- Performance management functions,
- Security management functions and
- Supervisory functions,

in order to enable collaborative network management among nodes of a self-managing network.

## 4.4 Resilience

Resilience and survivability is regarded critical to the future of our network infrastructures. Resilience is the ability of the network to provide and maintain an acceptable level of service in the face of various challenges to normal operation. We aim to engineer a system to protect itself from these challenges and to recognize the impact autonomously if the defence could not isolate the effects. In this case the system services must self-organize themselves to remain accessible whenever possible and degrade gracefully when necessary. As soon as the challenge ended the system must automatically and rapidly recover from degradation to normal operation. To improve future operation of a resilient system, it has to learn from past incidents and refine its operational and defensive mechanism.

On the other hand self-organizing and autonomic behaviour is necessary for network resilience that is highly reactive with minimal human intervention (see resilience principle 11). A resilient network must initialize and operate itself with minimal human configuration, management, and intervention. Ideally human intervention should be limited to that desired based on high-level operational policy. The phases of autonomic resilient networking consist of:

- initialization – auto-configuration of network components and their self-organization into a network
- steady-state normal operation – self-managing with minimal human interaction dictated by policy and self-optimizing to dynamic network conditions
- steady-state expecting faults and challenges – self-diagnosing and self-repair

This principle is one of a set of principles and four axioms introduced by the resilience framework. Notably, the axioms are:

A0. **Inevitability** of Faults

A1. **Understand** Normal Operations

A2. **Expect** Adverse Events and Conditions

A3. **Respond** to the Adverse Events and Conditions

which are supported by a set of principles:

P1. **Service Requirements** determine the need for network resilience

P2. **Normal Behaviour** must be specified, verified, and refined through monitoring to understand normal operations

P3. **Threat and Challenge** Models are essential to understanding and detecting potential adverse events and conditions

P4. **Metrics** are needed to measure and engineer network resilience

P5. **Resource Tradeoffs** determine the deployment of resilience mechanisms

P6. **Complexity** of the network in general, and resilience in particular, must be reduced to maximise overall resilience

P7. **Multilevel Resilience** is needed with respect to protocol layer, protocol plane, and hierarchical network organisation

P8. **Translucency** is needed to control the degree of abstraction vs. the visibility between levels

P9. **Trust and Policy** relationships are necessary in multi-user, multi-provider networks; resilience mechanisms must consider this tussle

P10. **Redundancy** in space and time increases resilience against faults and some challenges

P11. **Diversity** in space, time, medium, and mechanism increases resilience against challenges to particular choices

P12. **Self-Organising and Autonomic** behaviour is necessary for network resilience that is highly reactive with minimal human intervention

P13. **Security and Self-Protection** is an essential property of entities to defend against challenges in a resilient network

P14. **State Management** is an essential aspect of networks in general, and resilience mechanisms in particular; the alternatives of how to distribute and manage this state are critical to resilience

P15. **Connectivity and Association** among communicating entities should be maintained when possible, but information flow should still take place even when a stable end-to-end path does not exit

P16. **Context Awareness** is necessary for network components to operate autonomously to detect challenges

P17. **Adaptability** to the network environment is essential for a node in a resilient network to detect, remediate, and recover from challenges

P18. **Evolvability** is needed to refine future behaviour to improve the response to challenges, as well as for the network architecture and protocols to respond to emerging threats and application demands

Based on these axioms and principles we investigated current network systems from a service-oriented point of view. These service descriptions can then be tested against possible threats which can affect the network system.

The result of this investigation is our six-step two-phase strategy *D2R2+DR* for resilient network systems:

1. Real-Time Control Loop – *D2R2*:
    S1. ***D*efend** against challenges and threats to normal operation
    S2. ***D*etect** when an adverse event or condition has occurred
    S3. ***R*emediate** the effects of the adverse event or condition to minimise the impact
    S4. ***R*ecover** to original and normal operations, including control and management of the network
2. Background Diagnosis and Refinement – *DR*:
    S5. ***D*iagnose** the fault which has been the root cause of an error or failure
    S6. ***R*efine** behaviour for the future based on past D2R3 cycles

The integration of this strategy into the ANA architecture will be one of the challenges we are going to face in the ongoing research. To start this process, an initial design of components and interfaces to build a resilient network system has been developed and several open issues which must be further investigated have been identified.

# 5 ANA NODE

## 5.1 Introduction and scope

The content of this section mainly results from the work being carried out in tasks 1.3 (Network abstractions and communication paradigms) and 1.4 (Communication mechanisms) of work package 1, and task 4.1 (Testbed prototyping, management, and evaluation) of work package 4. While section 3 of this document provides a conceptual description of ANA and its operation, this section focuses on the description of the future implementation of ANA and how the abstract mechanisms previously presented are provided by "the low-level machinery" of the architecture. The objective of the two sections (3 and 5) is to provide a complete picture of ANA, from the conceptual abstractions and mechanisms down to the specification of the APIs offered at the implementation level.

This section is organised as follows. Subsection 5.2 provides an overview of the functionalities of the ANA node and introduces its main components. These components are then described in more details in subsection 5.3, and subsection 5.4 finally introduces the set of APIs that will be provided by the architecture.

## 5.2 Overview of the ANA Node functionalities

The ANA Node is a collection of both mandatory and optional software components that basically allow "to run ANA" on a physical device (e.g., computer, sensor, network processor, switch, router, etc). Three components can be identified:

- MINMEX, a minimal component for configuring and re-configuring network functionality.
- ANA Playground, where optional functionality is made accessible.
- ANA hardware abstraction layer.

Each of these elements, also shown in Figure 5-1, is briefly introduced in this section.

The mandatory and underpinning component called MINMEX provides the basic low-level functionalities which are required to bootstrap and run ANA. It also provides the generic set of methods (APIs) that are used by "clients" of the MINMEX (i.e. protocols, applications) to interoperate with MINMEX and the elements it hosts and with other clients of the architecture. The MINMEX is described in details in section 5.3.

The ANA Node also implements an abstraction layer which offers a generic access to the hardware upon which it is executed. In particular, the abstraction layer has built-in functions to access network interface cards either via the hosting operating system (e.g.,

Linux) or via dedicated drivers on e.g., embedded devices like network processors. To support legacy applications, an ANA Node can also provide an adaptation layer in environments that support e.g., Internet-style applications using the standard socket programming APIs. This allows re-using existing applications in ANA without having to modify their code and re-compile them. For applications specifically developed for ANA, the ANA Node provides a set of generic APIs that gives full access to all the functionalities of the ANA architecture.

Beside these two compulsory components, the ANA node provides a development framework (i.e. another set of APIs) coupled with a dedicated execution environment where the more elaborated and complex networking functionalities of ANA are placed. This flexible component, known as the "Playground", can host commodity and potentially generic networking elements such as monitoring probes, routing protocols, and cryptographical services, up to self-contained network stacks, generic peer-to-peer middleware, and complex and dedicated learning algorithms. By design, the ANA Playground hosts the autonomic protocols that will exhibit a self-* behaviour.

Note that in contrast to functional blocks (FBs), information channels (ICs), and information dispatch points (IDPs) introduced in section 3.1, the ANA node is not an abstraction of ANA. The ANA node is rather defined as the "recipient" that hosts fundamental elements: that is, the "node" is a property ("item x IS ON node y"), not an element that we need to define as an abstraction in our architecture. Only indirectly, the "ANA node property" is also captured through the compartment concept, wherefore it is also called the "ANA node compartment" (see section 3.2). By design, it becomes possible to query an ANA node in order to learn not only what high-level functions and services it hosts but also what network compartments it is connected to. This opens the way to offer choice to application on how to perform a communication task, while today an application *hard-codes* the network it wants to connect to (e.g., via well known protocol values such as AF_INET or AF_INET6 in socket programming).

## 5.3 ANA Node: architecture and components

The figure below is a conceptual representation of an ANA node whose most prominent features are the **MINMEX** area (Minimal INfrastructure for Maximal EXtensibility) and the **Playground**. The MINMEX area defines the common denominator among ANA nodes, and MUST be present in all implementations. The Playground hosts the optional protocols that one is free to develop with the help of the functionalities provided by the MINMEX elements. Note that although the Playground strictly contains optional components, a typical ANA implementation will be shipped with a set of "standard" features to permit access to basic network compartments.

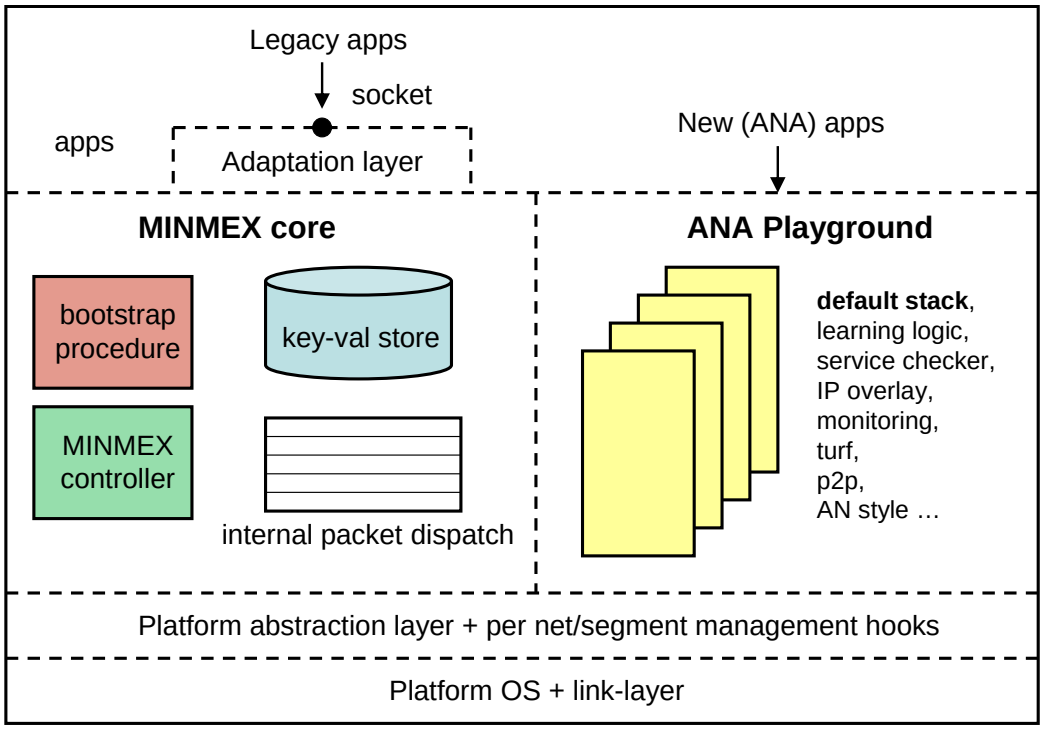



**Figure 5-1. The three components of an ANA node, embedded in an OS context.**

The "MINMEX approach" followed in the design of ANA permits to focus on a simplified core and declare richer and more complex functionality as *prolongations* of the architecture. In a sense, MINMEX forms the constant part of an ever evolving network: it is more an architecture framework than a network architecture by itself. Thus, a central role of MINMEX is to provide the generic functionalities to instantiate and run network compartments code and to provide a dedicated execution environment for each "instance" of a network compartment, should a node support many instances.

The operations offered by the MINMEX can be characterized as follows:

- Means to forward packets between applications, functional blocks, and interfaces.
- A basic directory service allowing access to ANA communication mechanisms, protocols, and compartments.
- A bootstrap procedure used to sense the network environment and activate compartments from the Playground.
- A MINMEX controller which performs a continuous assessment of the basic operation of an ANA node.

In the following subsections 5.3.1 to 5.3.4 we provide a more detailed description of these four basic areas covered by MINMEX.

### 5.3.1 The information dispatch framework (IDF)

As largely introduced in section 3.1 of the Blueprint, the distribution of data packets in ANA is achieved via IDPs (Information Dispatch Points). To briefly recap, data packets are always sent to an IDP which is itself bound to a functional block (FB) or an information channel (IC) (at least conceptually). This approach provides ANA with powerful indirection capabilities that permit to seamlessly redirected packet flows according to network internal needs without the sender having to become aware of any change: it will still send data to the same (start) IDP.

As shown by the set of examples of the node compartment section (3.2), data following a communication path is typically sent to a start IDP and then passed on through a chain of functional blocks to the downstream IDPs. This process is somehow similar to the hop-by-hop forwarding of data in today's Internet. However while in the Internet a "hop" in the forwarding process is typically a network hop (i.e. data is passed on to a neighbouring network node), ANA extends and generalizes this concept to data being forwarded inside a node among FBs and to other (network-wide) nodes via ICs. Note that while (strictly speaking) data is sent to an IDP, it is up to the entity bound to the IDP to know to which "next hop IDP(s)" the data should be forwarded. Hence while conceptually IDPs are powerful abstractions of ANA, implementation-wise an IDP is actually an entry in a functionally enriched forwarding information base (FIB).

The forwarding procedure inside ANA is illustrated in the Figure 5-2, which shows data being 1) received by a node compartment, 2) forwarded inside the node compartment and 3) forwarded to a distant node via an information channel. Data is received by IDP *b* which is bound to FB *f2* that has next-hop IDP *c*. Data dispatching carries on until reaching IDP *e* which calls a low-level function provided by the ANA Node abstraction layer that sends data via a network interface. Typically for link-layer forwarding, IDP *e* will store the information (e.g., MAC address) needed to reach the next hop network node. Note that the figure also provides an abstract representation of the forwarding table being used.

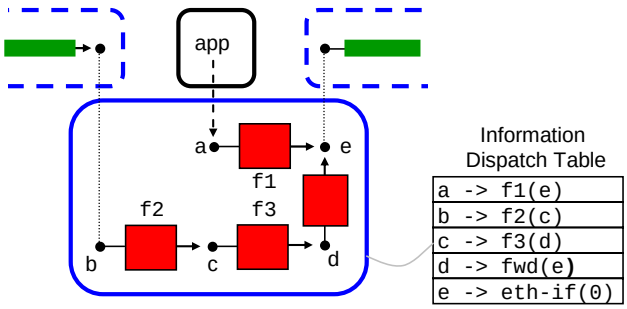



**Figure 5-2. Data dispatch inside ANA.**

In practice and as described in section 5.4, the interaction with the information dispatch framework is achieved via dedicated APIs: at the conceptual level one manipulates IDPs and has no direct access to the (real) internal data structure. With these APIs the (local)

instance of a network compartment can create and delete IDPs and set per-IDP mandatory information such as the function bound to the IDP and the next hop IDP.

A critical feature of the information dispatch framework (IDF) is that it is fully address and name agnostic: IDPs are identified by local labels (i.e. numbers) which have no networking meaning whatsoever. This is a key requirement since the IDF must dispatch data that (conceptually) belong to radically different compartments with any sorts of addressing and naming schemes. This *neutral* position allows this core component to support any new addressing or naming scheme that can be developed in the future.

To separate concerns, IDPs are stored in Information Dispatch Tables (IDTs). An IDT typically contains all the (node local) IDPs that belong to some communication context, e.g., a given compartment or a given view of the node compartment. This conceptual separation prevents, e.g., unauthorized manipulations of IDPs or excessive resource (e.g., memory) consumption by misbehaving or faulty components. Note that implementation-wise there may only be one table where each entry (IDP) would contain a field indicating which IDT it belongs to.

To conclude this subsection, it is worth reminding that while the information dispatch framework provides flexible and powerful mechanisms to create and dynamically modify data paths, it is not responsible for setting up and changing these paths. That is, the "control" is typically left to higher level entities such as the functional composition and monitoring frameworks, native ANA applications, and functional blocks.

### 5.3.2 Key-Val Repository

Although conceptually a compartment contains a database where membership relationship is stored, outside clients cannot access this database directly, in general. An ANA node compartment, however, makes this database partially accessible through a key-val repository. Third parties can create entries with self-chosen values which then later can be retrieved via the standard "resolve" functionality.

As an example, we envisage (physical) nodes in an ANA network to announce available services to neighbours by depositing entries in the neighbour's key-val repository. Local clients will use the resolution process to discover these services and to obtain access to them via a local IDP. Behind the local IDP, the remote node has made sure that service requests are forwarded appropriately to where the service request will be worked on.

In a first phase, a simple tuple storage is provided, akin to a MIB with (one level of) key-value associations. This key-val store will be used for system variables, binding of method names to the function and scratch memory for coordination among extensions, for example. Different policies for adding, extracting and erasing entries will be explored: As we gain more experience with the usage and the requirements of ANA-aware networking functionality, a more sophisticated database interface will be proposed.

### 5.3.3 The bootstrap procedure (BP)

The bootstrap procedure (BP) is a set of generic mechanisms that must be supported by all implementations of ANA. It is used by an ANA node to discover its immediate surrounding network environments such as e.g., neighbouring nodes and active and reachable compartments. Since the bootstrap procedure is the first network component of ANA to be executed, it cannot rely or use data paths created and maintained by "production" protocols. This hence requires that the operation of the BP is fully independent.

In practice, the bootstrap procedure may initially rely on a simple (broadcast based) discovery phase which will generate entries in the key-val repository. Note that the bootstrap protocol may also be configured to kick-start a more complex service discovery framework that would be executed in the Playground (see deliverable D2.1 [5] for a more detailed of service discovery in ANA). However, all ANA implementations must support the generic mechanisms in order to guarantee the presence of a least common denominator.

In addition to discovering its neighbourhood, the BP also loads the static configuration of the ANA node. This configuration is typically maintained by human users and specifies which compartments should be instantiated upon start-up and if a compartment should be used by default. It may also instantiate functional blocks and information dispatch points and activate protocols that reside in the Playground.

### 5.3.4 The MINMEX Controller (MC)

The MINMEX controller (MC) performs a continuous assessment of the basic operation of an ANA node. It is truly autonomic and basically performs a sanity and health check of the components running inside the node. The core objective of the MINMEX controller is basically to "protect" the MINMEX elements from faulty or misbehaving components in order to guarantee the performance of the ANA subsystem. Keeping such control separate and not embedded in all the other elements is beneficial for two main reasons. First, it simplifies the design and code of the other MINMEX elements and second, it allows control operations to be centralised and coordinated in one entity. The main drawback is that all control relies on a single entity which becomes a very sensitive component of the architecture. However, the MINMEX controller could potentially be implemented in a distributed way which would greatly reduce the risks of a total unrecoverable failure.

The rest of this section outlines some of the fundamental activities of the MINMEX controller (MC). This description is not exhaustive and additional tasks may be added in the future as needs appear. However, note that the following control operations are mandatory and must be supported by all implementations of ANA.

- The MC periodically controls all the forwarding paths and states of the information dispatch tables. In particular, it must detect forwarding loops which may have been accidentally created. If a loop is detected, the MC must prevent entities to send data on the loop: this interfering action must generate an

appropriate error code back at the sender side. Data circulating inside a loop must be destroyed.

- The MC performs a garbage collection of unused or expired IDPs that may have become orphan IDPs if the entities using them are malfunctioning or have crashed. Such IDPs must be removed from the system in order to free resources. Note that this verification may become redundant or obsolete if IDPs are stored in a soft state manner.
- In a similar manner, the MC performs a garbage collection of the entries of the key-val repository. This task may also become redundant or obsolete if the entire repository relies on soft state storage.
- In addition, the MC periodically controls the state of all the active functional blocks running in the system. If the MC detects that a functional block is malfunctioning or crashed (via e.g., the failure of a polling mechanism), the MC must either remove the FB from the system or try to re-instantiate this functionality. The action to be taken in the case of a failure may be specified by the FB itself upon start-up.
- The MC provides the lowest level of access rights and control for accessing information dispatch tables. For example and as already described, MINMEX may maintain separate IDTs for different communication contexts: in that case, the MC must prevent un-authorised access to IDTs and return an appropriate error code to the entity performing the rejected operation.

### 5.3.5 The Playground

The Playground contains all the optional extensions of the base ANA node. This is where complex protocols and functions are implemented and where network-wide autonomicity is achieved. It is the area where variety will exist, perhaps even programmability in an active networking sense. Note that in the remainder of this section we intentionally avoid using the term “functional block” to refer to protocols and functions residing in the Playground: an FB may indeed contain multiple functions or a given functionality may be implemented in a distributed way by multiple FBs. We hence prefer to use a more neutral vocabulary.

Typically, the Playground will host both commodity and dedicated functionalities. Commodity elements are “public” components that one can re-use in order for example to compose more complex functions in an on-demand manner. This includes, e.g., cryptographical primitives, compression schemes, queuing systems, reliable transmission schemes, generic learning algorithms, and error recovery codes. How such elements can be dynamically composed is described in deliverable D2.2 [6].

The Playground may also contain complex components that are specialised for a certain task and can hardly be re-used outside their initial design space. This category includes, e.g., fully-contained network stacks/heaps (”compartments code”), highly dedicated and optimized protocols such as delay-tolerant transmission schemes, and schemes for network coding. This classification is highly subjective and does not exist inside the

Playground: we here merely want to highlight that functions inside the Playground range from simple re-usable components to complex and highly specialized elements.

Note that in practice, the protocols of the Playground interact with the MINMEX elements via specific APIs that allow them to, e.g., initially communicate with the node compartment and then with network compartments, instantiate functional blocks and information dispatch points, and create information channels to distant peers.

## 5.4 Implementation Details

This section shows in more details the generic API for ANA.network compartments. It serves mostly as an internal technical development document and is a prerequisite for implementing the first ANA prototype. We envisage that this API will undergo substantial changes; however, it is also important to document it already now in order to help readers understand high level concepts introduced earlier in the blueprint, by permitting to relate the high level concepts almost to bits and bytes. Part of this API will, in a later phase, form the basis to write RFC-style documents enabling any third party to implement an interoperable version of ANA.

Positioning of this API: We envisage that an application, which wishes to use the functionality of an ANA network compartment, will start in a first place outside this compartment. It has to formally connect to it, whereafter it can provide services through it, or simply use the compartment's services. The API is at the border of the execution environment the application is running in. Below this API, there can be a shim interface directly to some OS kernel, or it can be a user space implementation of a full ANA node, or proxying code that connects to a compartment on a remote ANA node, in form of a library, or classes etc.

The ANA API is subdivided in different packages, or levels, that rely on each other.

- *API Level 2 (AL2)*: provides access to ANA functionality with transparent method invocation or remote procedure calls
- *API Level 1 (AL1)*: provides access to ANA "first order citizens" with typed message exchange.
- *API Level 0 (AL0)*: provides access to ANA low level functionality, offering raw datagram exchange.
- *API Level -1*: offers a choice of different connectivity technologies.

These levels express different degrees of sophistication and do not imply a layering as in networking. If an ANA application wishes, it could be fully based on level 0 API, in the same way an X-Window application could be written at the level of pixels, instead of widgets.

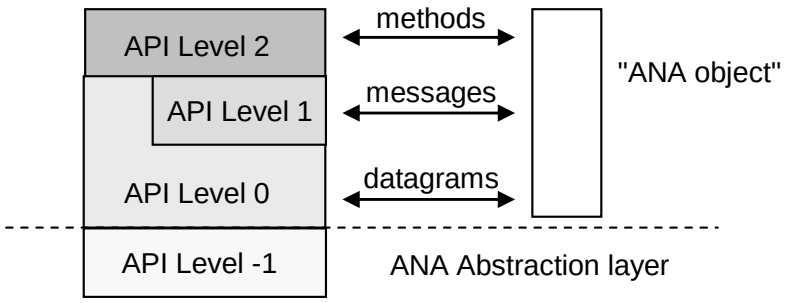



**Figure 5-3. API Levels in ANA.**

In the following we introduce for each API level a list of initial functions or methods as we see them at a time point before continuing on the prototype implementation.

### 5.4.1 Level 2 API

At API level 2, a programmer sees ANA in terms of ordinary objects. Functionality can be accessed through ordinary method calls, object destruction automatically runs the necessary low level cleanup actions. In the following, the API is expressed in a mixed style of Java and C++ language.

```
class anaCompartment {
  anaCompartment(String Description_of_compartment);
  ~anaCompartment();
  getDescription();

  /* publishes a service in the compartment */
  publish(String context, String service_descr);

  /* unpublishes a service from the compartment */
  unpublish(String service_descr);

  /* resolves a compartment service, returns a channel */
  resolve(String context, String service_descr, char chanType, String my_descr);

  /* lookup checks the availability of a service */
  lookup(String context, String service_descr, String my_descr);    …
}

class anaIC {
  send(void *data);
  getDescription();
}

class anaFB {
  getDescription();
}
```

These class definitions are only for demonstration purposes and are not indicative for the number and type of methods.

### 5.4.2 Level 1 API

At API level 1, we provide a library of procedures for interacting with ANA compartments, information channels and functional blocks by sending control protocol messages. This library also contains the logic for carrying out the dialogue (request/reply patterns etc) with some remote "object".

The AL1 procedure does not remove the need to establish and manage communication with an ANA node at AL0. But once attached (using AL0 procedures), an application can use the rest of the functions of this level to easily talk to ANA objects through IDPs, e.g., having ready-made procedures for creating requests and parsing result messages etc.

We envisage that each ANA object provides for its offered methods two procedures for "external representation handling".

- `encode function :` builds the control message in the right format
- `decode function` : parses the formatted reply and extracts the needed information.

In this section we use a C style language for writing down the API:

```
#ifndef _ANA_AL1_H
#define _ANA_AL1_H

#include "anaLib0.h"

int anaL1_requestReply(anaLabel_t destLabel,
                   void *data, unsigned short len,
                   anaCallback_t fct, void *aux );
/* The requestReply function will send the given data,
   i.e. 'len' Bytes, to the given destination label.
   Beforehand and internally, a "return IDP" is created
   and bound to the indicated function fct. This function
   will receive an upcall once the result is ready.
   Returns 0 on success, a negative value otherwise.
*/

// Below we provide an example of two formatting functions
// that a client might use when talking with an ANA compartments.
// These functions will only build a correct control or result
// message; The sending is done with requestReply function
// described above .

int anaL1_encLookup(void **message, char *context, char *service, char *myDescription);
/* This function creates a lookup message (filled in message pointer)
   of the given id value. The function returns the size in bytes of the encoded message
   Returns 0 on success, a negative number otherwise.
```

```
*/

int anaL1_decResponse(void *replyMsg, char *fieldName, void** val);
/* This method parses a reply message from a lookup() request
   and returns the found value (to be selected by the field's
   name). On success, the val pointer will point to the value of the requested field.,
   Returns the size of the requested field on success, a negative number otherwise.
*/

#endif
```

The following figure lists the AL1 functions for the time being. Please note that the exhaustive list of control functions proposed is not definitive, it will most certainly be modified during the development process.

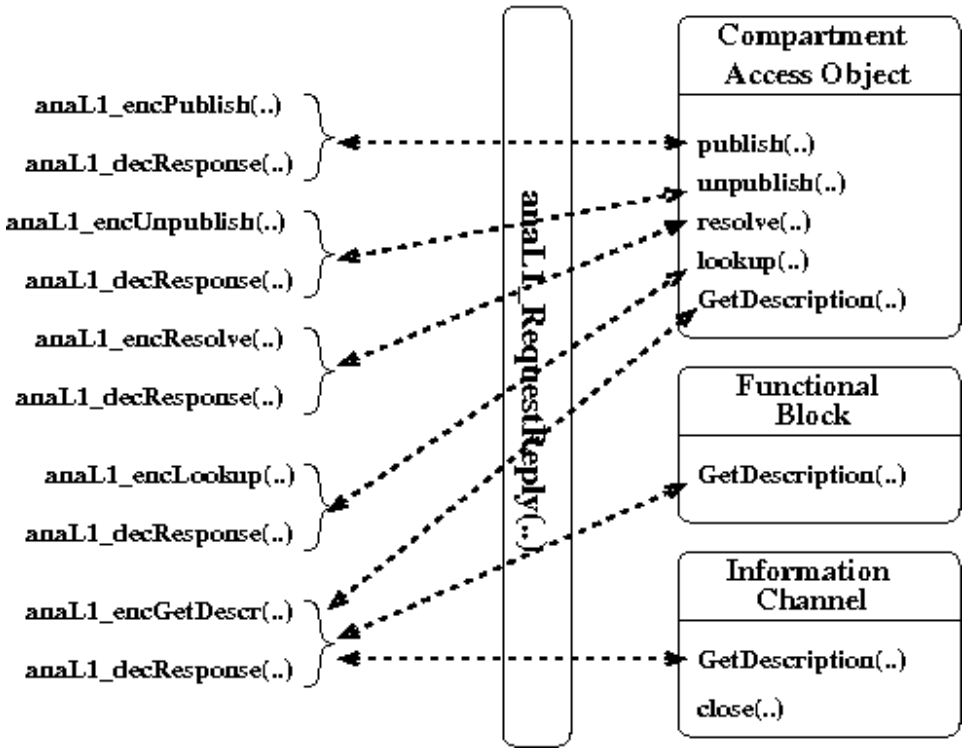



**Figure 5-4. API level 1 Functions.**

## 5.4.3 Level 0 API

In Level 0 we group all basic functions that permit to access an ANA node in a platform unspecific way.

```
#ifndef _ANA_AL0_H
#define _ANA_AL0_H

typedef int           anaHandle_t; // <0 is an invalid handle
typedef void * anaLabel_t;  // IDP identifier, NULL is an invalid label
```

```
typedef void  (anaCallback_t)(void *data, int len, void *aux);

int anaL0_send(anaLabel_t lab, void *data, unsigned short len);
/* Sends a data packet of length "len" to the information
   dispatch point labeled with "lab".
   Returns number of bytes sent on success. Note that this is an unreliable
   transmission (i.e. the packet can be lost despite a success
   return code).
*/

anaLabel_t anaL0_registerCallback(anaCallback_t fct, void *aux, anaHande_t viewer, int
permanent);
/* Requests the ANA node compartment to create a callback information dispatch point,
returns the chosen label. The viewer argument specifies in which IDT view this IDP
should be created.
Any packet sent to this IDP will result in an upcall to "fct", also passing the "aux"
parameter beside the actual packet.
   Note that the transmission is unreliable i.e., packets
   sent to the IDP may never lead to calling "fct".
   Returns 0 on success, a negative value otherwise.
*/

int anaL0_unregisterCallback(anaLabel_t label);
/* Unbinds the callback that is bound to the given label.
   The label and any associated resources are freed.
   Returns 0 on success, a negative value otherwise.
*/

#endif
```

### 5.4.4 Level -1 API

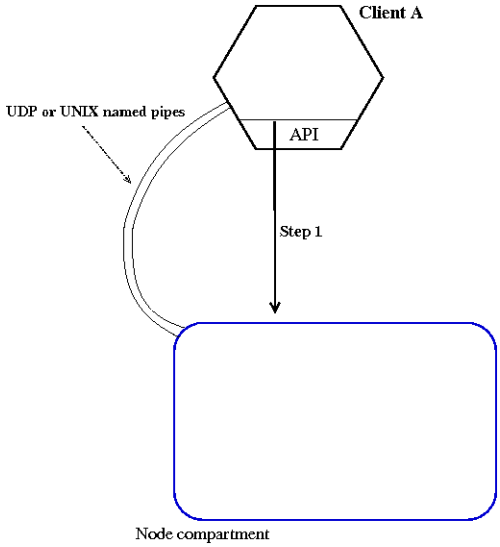
The -1 level defines the platform dependant initial communication mechanisms between the client and the ANA node. It is used to attach a client to the ANA world. The communication mechanisms used in this level are by then exterior to the ANA world, which is reflected by the negative index of this API‘s level.

Being platform dependant by nature, this API is not specified in this document and is dependant on the development process. Communication between the client and the ANA node at this level can be implemented in different ways (e.g., named pipes and UDP sockets).

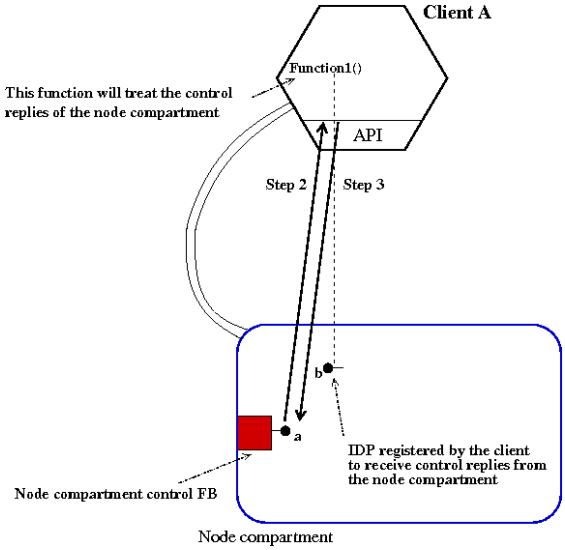
### 5.4.5 Example Scenario

In this scenario a client A wishes to communicate with a client B present in the same LAN. The needed steps for this communication are detailed below:

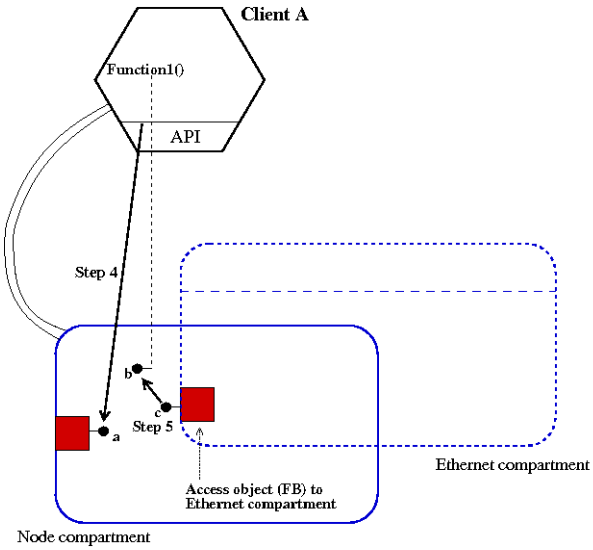
- *Step 1*: Client *A* attaches to the default ANA Node Compartment on it’s runtime system



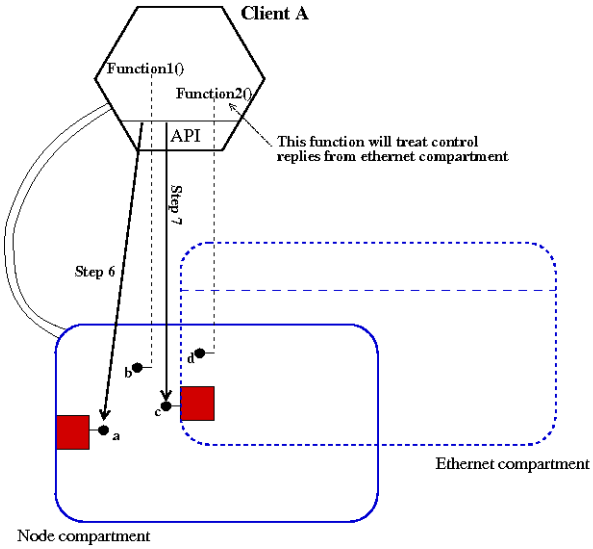

- *Step 2*: Client *A* receives in return an IDP label *a* for control communication with the ANA node compartment
- *Step 3*: Client *A* registers an IDP label *b* for future result messages and maps it to the treating function *Function1*



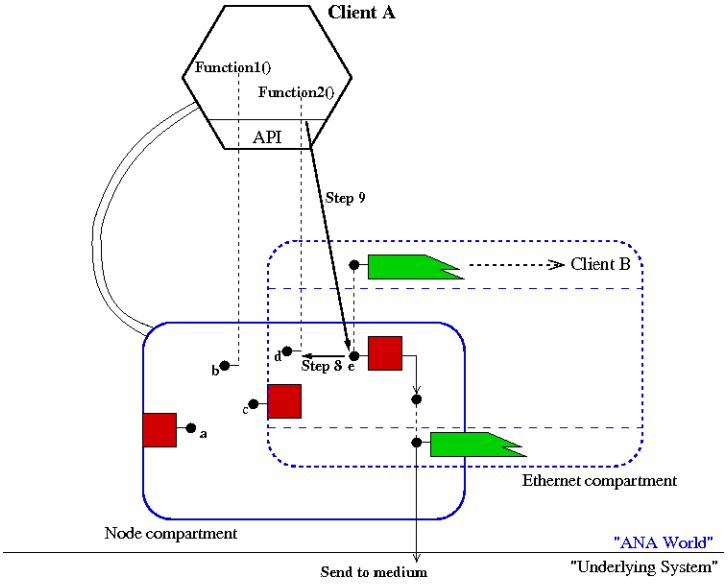

- *Step 4*: Client *A* then asks the node compartment to instantiate the Ethernet compartment
- *Step 5*: Client *A* receives an IDP label *c* for control communication with the Ethernet compartment




- *Step 6*: Client *A* then registers an IDP label *d* for future control responses of the Ethernet compartment and maps it to the treating function *Function2*
- *Step 7*: Client *A* then requests the Ethernet compartment to resolve a service offered by a client *B*




- *Step 8*: Client *A* receives in return the IDP label *e* to communicate with client *B*
- *Step 9*: Client *A* sends a “Hello world " message to *B* through IDP *e*




## 5.4.6 Code samples for levels 2, 1 and 0

In this section we demonstrate the use of the ANA API AL0 to AL2 for implementing the scenario described in the previous section.

### *Programming at ANA API Level 2:*

```
main() {
   anaCompartment node = new anaCompartment();
   /* covers steps 1, 2, 3*/
   anaCompartment EthCompartment = node.resolve(“.”, "eth-access-0", ‘u’, NULL);
   /* covers steps 4, 5, 6*/
   anaIC chann = EthCompartment.resolve("*", “client B description”, ‘u’, “clientA”);
   /* covers steps 7, 8*/
   chann.send( "Hello World!");
   /* step 9 */
}
```

### *Programming at ANA API Level 1:*

```
#include <stdio.h>
#include <stdlib.h>
#include "anaLib0.h"
#include "anaLib1.h"


static void ethReplyUpcall(void *reply, int len , void *aux)
{
     anaLabel_t infoChannLabel;
     char *msg = "Hello World!";
```

```
    anaL1_decResponse(reply, "label", &infoChannLabel);
    /* step 8 done*/
    anaL0_send(infoChannLabel, msg, strlen(msg)+1);
}

static void nodeReplyUpcall(void *reply, int len , void *aux)
{
    anaLabel_t ethCompartment;
    void *ethLookupRequest;
    int dummy,;

    anaL1_decResponse(reply, "label", &ethCompartment,);
    /* step 5*/
    requestLen  =  anaL1_encLookup(&ethLookupRequest,  "00:11:22:33:44:55",,"client  B
descr", "client A" );

    requestReply(ethCompartment, ethLookupRequest, requestLen, ethReplyUpcall, 0);
    /* steps 6 and 7 */
}

int brick_start()
{
    anaControlLink_t ctl;
    void *nodeRequest;
    int requestLen;

    /* steps 1 and 2 are performed by the library*/
    anaLabel_t nodeCompLabel = anaL0_getNodeCompLabel();

    requestLen = anaL1_encResolve(&nodeRequest, ".", "eth-access-0", 'u', NULL);

    anaL1_requestReply(NodeCompLabel, nodeRequest,  requestLen, nodeReplyUpcall, 0);
    /*  steps 3 and 4*/

    return 0;
}
```

### *Programming at ANA API Level 0:*

(In the following code sample we assume that the control messages are encoded as ASCIIZ strings).

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "anaLib0.h"

static anaLabel_t nodeSendLabel;
static void fail(char *reason){
    fprintf(stderr, "demo failed: %s\n", reason);
    exit(-1);
}
```

```
static void ethCtlMessageUpcall(void *data, int len, void *aux)
{
     anaLabel_t infoChan;
     char *msg = "hello ANA world!";

     /* step 8 : we extract from the data the label of the
        information channel, store in  infoChan Label */
     sscanf((char*) data, "%s", &infoChan);

     if (anaL0_send(infoChan, msg, strlen(msg)+1))
            fail("could not send packet to information channel");
     /* step 9 done*/
}

static void nodeCtlMessageUpcall (void *data, int len, void *aux)
{
     char cmd[1000];
     anaLabel_t ethLabel;
     anaLabel_t ethReplyLabel;

     /* step 5 : we extract from the data the label of the
        Ethernet compartment */
     sscanf((char*) data, "%s", &ethLabel);

     ethReplyLabel = anaL0_registerCallback(ethCtlMessageUpcall, NULL, NULL, 0);
     if (ethReplyLabel == 0)
            fail("could not register callback");
     /* step 6 done*/

     sprintf(cmd, "resolve id=00:11:22:33:44:55 return=%s)", ethReplyLabel);
     if (anaL0_send(ethLabel, cmd, strlen(cmd)+1))
            fail("could not send packet to Ethernet compartment");
     /* step 7 done*/
}

int brick_start ()
{
     anaLabel_t     nodeRecvLabel;
     char           cmd[1000];
     anaLabel_t     nodeCompLabel = anaL0_getNodeCompLabel();

     /* step 1 and 2 done by the library*/

     // the client registers an IDP for receiving
     // results from the ANA node compartment
     nodeRecvLabel = anaL0_registerCallback(nodeCtlMessageUpcall, NULL, NULL, 0);
     if (nodeRecvLabel == 0)
            fail("could not register callback");
     /* step 3 done*/

     sprintf(cmd, "resolve id=eth-access-0 return=%s", nodeRecvLabel);
     if (anaL0_send(nodeSendLabel, (unsigned char*) cmd, strlen(cmd)+1))
```

```
            fail("could not send packet to node compartment");
        /* step 4 done*/

        return 0;
    }
```

# 6 INFORMATION MANAGEMENT

Achieving self-* properties requires more than just acquiring raw data about the operating environment and state. This data has to be used, i.e., transformed into *information* that can be effectively used to achieve these properties. For instance, self-configuration requires that an entity knows its configuration, that it knows the components and resources which are available for potential adaptation, and that it can reason about the impact of configuration changes. Furthermore, self-healing requires that a system is able to define or to learn what the normal condition is and compare it with monitoring results in order to recognize deviations from the normal condition. The capacity to proactively circumvent issues that could cause service disruptions means that the system must be able to perform a retrospective analysis after a service disruption, i.e., to study data from the past in order to identify which sequence of events might lead to a service disruption.

In many cases a single source of data and information is not sufficient. Therefore, it is important for the ANA architecture to develop proper abstractions for providing information and sharing information between entities. This information can be derived from monitoring data or it is part of a description of an entity. Such a description could be a describing the entities configuration, available components, available resources, etc. Furthermore, the ANA architecture must provide means to persistently store data and information, because for certain tasks, like the retrospective analysis in self-healing it is necessary to study past data. Obviously, some kind of syntax has to be used for these descriptions and a certain data model has to be used for handling data, like monitoring data.

In summary, information management is needed to address the following requirements:

- Need to exchange and use monitoring data and other types of metadata as the basic element of protocols and algorithms with self-* properties.
- Enable easier self-configuration and self-organization, e.g., through better service discovery solutions.
- Need better solutions for self-optimization, e.g., for the case of structuring communication into uni-cast, any-cast, and multi-cast communication for efficient resource utilization from networks viewpoint.

Please note that we have a different understanding of *information* in this section compared to *information in ICs* in the other sections of the ANA blueprint (this is due to the timely parallel development of the ANA concepts). Therefore, we will first explain the scope of *information management* in section 6.1, before we described the *information flow requirements* in section 6.2. In the following sections (remaining part of section 6), we present the *information flow concepts* (i.e., the ANA deliverable D1.6v1 "Information Flow Data Items and Interface specifications") that are needed for the information management.

## 6.1 Scope of Information Management

Achieving autonomic behaviour requires the exchange of information of different types. For instance, auto-configuration requires that an entity has information about its own configuration that it knows about components and resources which are potentially available for interaction and usage. Similarly, other self-* properties depend on the availability of information for the establishment of communication paths and optimization of network operation.

The data that is processed and handled in a communication network can be categorized into three classes, where each of these classes is defined by the particular way data is used.

- **Content Data**: relates to data that is stored or transferred by the network without need for interpretation. Content data is represented in ANA as a sequence of bits handled by a functional block (FB) or an information channel (IC).
- **Metadata**: is data that is used to describe other data objects.
- **(Network) Information**: is data that is used for decision making in autonomic networks. For that purpose, it is necessary for the decision making components to interpret it. In many cases, timely delivery of the required information is crucial to achieve sound decisions.

Figure 6-1 shows the resulting classification of data based on its usage in the domain of networking. In the figure, the root type "data" is abstract and the usage of the data item is written on each arrow that descends from the root. The main point of our classification of data is that the *type of a particular data item depends only on its usage.* Consequently, a single data item may belong to several classes, even at the same time, depending on how it is used. For example, a TCP packet header may be content for an Ethernet driver that only interprets Ethernet headers, signalling and control information for the TCP protocol, and monitoring information for a passive traffic analysis probe.

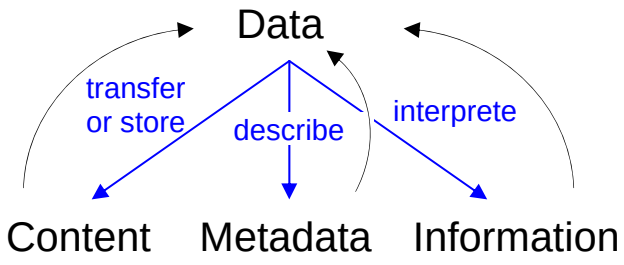



**Figure 6-1: Classifying Data**

One important part of this simple classification scheme of data is that it can be recursively applied, which is indicated in Figure 6-1 by the arrows that are drawn from content, information, and metadata back to data. For example, data in the class of metadata can be interpreted and used for decision making and thus classified as information, or information can be transferred via an information channel between two functional blocks without any interpretation and would then be regarded than as content.

The ***Information Flows*** are an important part of the ANA architecture which presents abstractions for providing information and sharing information between entities. It defines a collaborative presentation of network information and an extendable architecture to distribute information.

## 6.2 Information Flow Requirements

We have identified that the main goal of information flows is to provide a principal concept within the ANA architecture to enable functional blocks to exchange information to use it for decision making. In the ANA deliverable D1.2 "ANA Requirements" [21], we argue that the ANA architecture should be based on a minimal set of assumptions and required standards in order to make it future-proof. This argument needs also to be applied to the concept of information flows. Thus, the architecture should not describe which data can be used in which way for decision making and how it could or should be interpreted. Instead, a generic solution is necessary that allows functional blocks to dynamically explore how data can be used and for what purpose.

### *6.2.1 Information Hook*

An information flow includes at least two functional blocks, of which one has the role as information provider and the other as information consumer (respectively decision maker). Naturally, the information provider is the entity that could provide the information about which data it can provide and how it could be used. Thus, the information provider needs to provide a self-descriptive interface. This interface would serve as a hook to which the consumer can connect and exploit how to make use of the data. Such a hook should exist in each element of the ANA architecture.

This information hook needs to have two main properties:

1. ***Unified generic interface:*** The hook needs to have a unified interface so that each ANA element knows how to access it. Otherwise, information flow establishment is impossible. The interface needs to be generic and as simple as possible. In this way, the interface can be used in an extensible way: Imagine a simple interface that provides by request a more complex interface more specific to the particular information hook.
2. ***Self-describing:*** The hook needs to be able to describe what kind of information it can provide. In this way, any ANA entity can query the information hook of a particular entity in order to learn if that entity can provide the information that it needs. Without such a capability, the abstraction of providing information and sharing information between entities becomes pointless. The information flow architecture needs to consider what kind of syntax and semantics the descriptions should have. XML-type of self-describing document structures is a possible approach.

### 6.2.2 Information Flow Characteristics

We need to consider what determines a specific ***information flow type***. We observe three distinct characteristics that may vary:

***Participants:*** There can be two or more participants in an information flow. One or more participants are the sources, i.e. information providers and one or more participants are the destinations, i.e. information consumers.

***Information type:*** The information type plays an important role in defining a particular kind of information flow, e.g., monitoring, signalling and control, or configuration information.

***Non-functional requirements:*** Different types of information flows may have different constraints. The constraints include timing constraints and constraints related to the sensitivity of information (e.g., classified information). In many cases, the information type determines some of the constraints. It might be that for the same information type different constraints occur depending on the situation and scenario in which the information is needed, e.g., the same piece of information might be needed as regular notifications in one scenario whereas in another scenario it might be only required on demand.

Starting from the next section, we describe the concepts for ANA information flows based on the above identified requirements. The key concept is the information hook which we introduce and exemplify in sections 6.3.1 and 6.3.2. We also revisit the example scenario from section 5.4.5 on communication establishment from the information flow point of view in section 6.3.3.

## 6.3 Information Flow Concepts

ANA provides a new architecture for data and information transfer. Instead of providing a fixed network stack for communication, the required functional blocks are combined into a function chain on demand. The elements which can be composed, i.e., FBs, can consist of whole protocol functionality (e.g., TCP, UDP) or may provide only micro-protocol functions, like error control or encryption.

In this very dynamic scenario, the *Information Flow Framework* plays a fundamental role in the whole *function chain management* process. It is related to all information exchange aspects that effectively allow the *establishment*, *operation* and *optimization* of the function chain. We identify two clear objectives which information flow concepts address:

- Describe ANA entities in order to support an effective establishment of a generic function chain for data and/or information transfer: Due to the dynamicity of the ANA infrastructure, each ANA entity has to provide information about both its purpose (i.e. the services it provides) and how other entities can interact with it.

- Allow interactions between ANA entities in order to effectively use the services they offer to each other. To this aim an entity enables other entities to access its services through its interface.

Information processing itself is done by FBs and is therefore not part of the information flow framework, even if it is using information flows. For example, a monitoring FB that computes some aggregate metrics by combining several monitoring information flows is not an information flow itself, but it uses information flows to implement a specific kind of monitoring service. In other words, information flows are concerned with those processes that involve information exchange in ANA at different stages:

- Before a function chain for data/information transfer is established, in order to support the functional composition process;
- After establishing a function chain for data/information transfer, in order to support its management and exploitation.

There are two aspects of information flows that are described in the following sections, first the possibility to learn via an *information hook* about the service(s) an entity provides, and second, the use of the entity's services by invoking the corresponding functions/methods.

## *6.3.1 Information Hooks*

In order to specify information hooks, we first outline some basic assumptions about FBs and how they function, and then describe the generic information hook, and apply the concept of information hooks also to composed FBs and ICs.

### 6.3.1.1 Basic assumptions for FBs

FBs are entities that process information. In order to do this, typically information is fed into a FB and the result of the information processing is the corresponding output. This input and output is always done through the FB's interface. Without restricting ourselves to a particular implementation strategy for FB interfaces, we assume that it consists out of a set of functions. Figure 6-2 illustrates a FB with seven functions. The mandatory two functions that must be implemented by each FB are named *get_description()* and *send_description()* and compose the information hook of that FB (which we describe in detail in the next section).

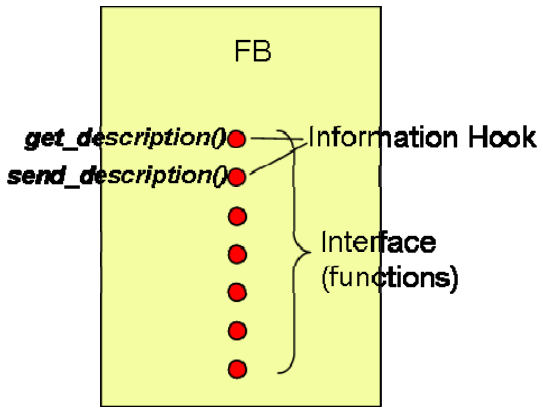



**Figure 6-2: Functional Block**

Figure 6-3 shows a FB with bindings to IDPs for incoming and outgoing data. The interaction with a FB happens through its functions. An IDP is the entry point of an IC and it is bound to a function of the FB. The figure shows three functions that are each bound to a separate IDP for incoming data. When data is received, the corresponding function is dispatched via the IDP. As shown in the figure, a function can be naturally bound to several different IDPs. Output data is sent through the ANA node internal packet dispatcher that takes care of forwarding the packet to the correct IDP. In general, the relationship between functions and IDPs is as follows: to each function $n \geq 0$ unidirectional IDPs can be bound.

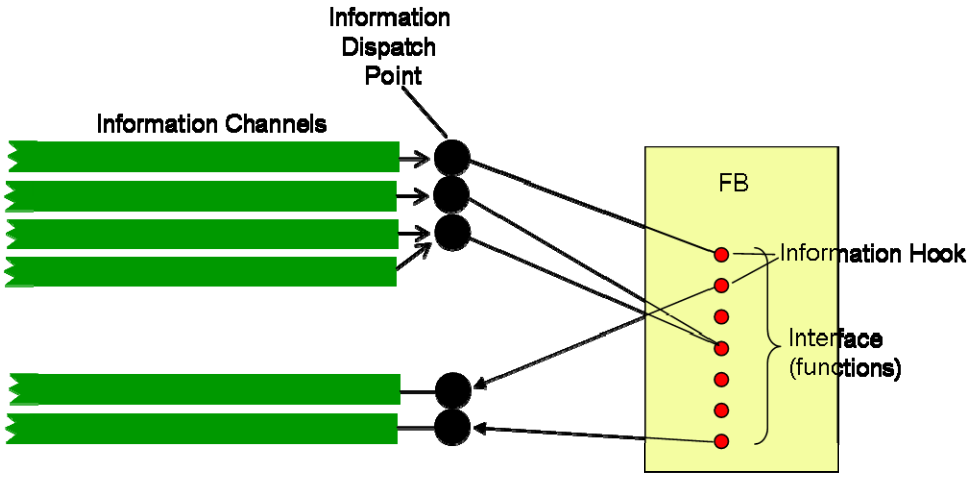



**Figure 6-3: Functional Block with Bindings**

### 6.3.1.2 Generic Information Hook

Each FB and IC in ANA that supports the information flow concept implements an information hook. This hook provides a description of the entity. The content, format, etc. of the description depends entirely on the entity itself, but the minimal mandatory description may be specified within the policies of a particular compartment. Basically, there are three interaction styles with ANA entities possible: (1) request/reply, (2) pure

input and pure output, and (3) all implemented by the entities functions. For example, an entity may allow other entities to access information about it, which is a request/reply interaction. Additionally, it might also allow other entities to control its behaviour, which is a pure input function. An example for a pure output function is the unsolicited announcement of the entity by broadcast. Independent of the interaction style and the particular functions that are implemented by ANA entities, the important principle for FBs, composed FBs and ICs is that all their functions are described by the *entity description*, including the syntax of the functions and the semantics.

As pointed out by the requirements analysis of section 6.2.2, the information hook should be generic and self-describing. A generic hook allows in principle any ANA entity to access and request a description of any other entity having an information hook. As a consequence, the generic hook is kept minimal: it provides access to the description of the entity. This description provides information about how to interact with the entity bearing that hook, i.e. in essence a description of its interface. The element accessing the hook would then learn what information and services the entity bearing that hook can provide and how they can be accessed, because the entity description comprises a description of the semantics of the functions, i.e., what they do, and the syntax of the functions, i.e., how to properly invoke them with the correct parameters, etc.

The question which language and tools to use for entity descriptions to (a) describe the functions and (b) interpret these descriptions is not part of the generic information hook. Instead it must be determined/standardized for a particular ANA compartment. An example for possible language and tool that might be used for the syntactical aspects of functions is the Interface Definition Language (IDL) [26] and a kind of stub compiler to generate the two procedures for external representation handling in the Level 1 API of MINMEX (see section 5.4.2). To describe the semantics of functions and their parameters, a combination of the Resource Description Format (RDF) [23] and XML as it is proposed in [25] could be used. We give detailed examples of these approaches in section 6.3.4.

The information hook is composed of two functions: *get_description()* and *send_description(). Get_description()* returns the entity description by invoking *send_description(),* which is illustrated in Figure 6-4. After receiving the description, the requesting element (e.g., FB) on the left side in Figure 6-5 knows how to interact with the element on the right side. Thus, afterwards the interaction happens through the element specific interface, as illustrated in Figure 6-5. The entity description type is mandated by the compartment policies, which ensures that the receiving element is able to interpret the description. While the typical interaction to receive an entity description would comprise request/reply style functions, as in MINMEX Level 1 API, the information flow concepts support also to use only *send_description()* to distribute without previous request the entity description.

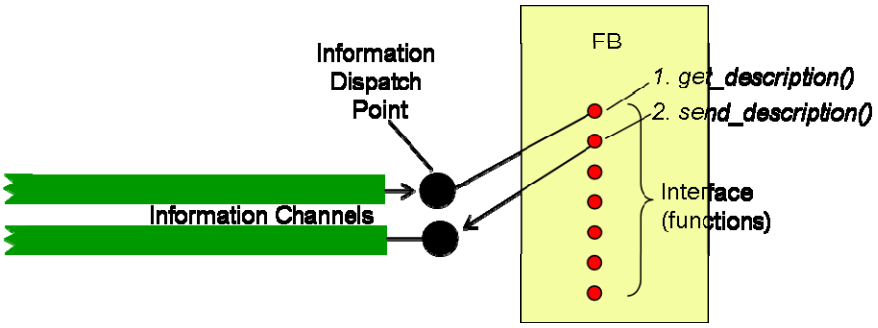



**Figure 6-4: First the interaction happens through the Information Hook**

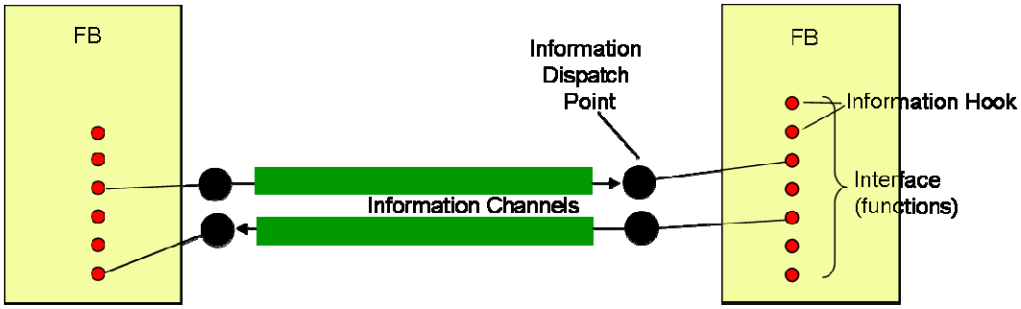



**Figure 6-5: After receiving description the interface can be accessed**

Each ANA entity has a *unique identifier* within a compartment. This identifier can be used to access the information hook in order to get a description of that entity, i.e. the *get_description()* function corresponding to a specific identifier is called. Such a procedure is often the starting point of communication. For example, a FB would like to get certain information from another FB whose identifier it has resolved. It may be that the requesting FB does not know how to access the information or does not even know whether such information is provided by the other FB. In such a case, the first step for the requesting FB is to invoke the *get_description()* function corresponding to the resolved identifier.

### 6.3.1.3 Composed FBs and ICs

Composed FBs and ICs are ANA entities that must implement an information hook as well. The fundamental difference between composed FBs and ICs is that ICs are made of FBs that reside on at least two different nodes. Describing composed FBs and ICs is more complex than describing a single FB, because these entities are composed of at least two FBs and they aim to represent more than just the sum of the individual FBs. Composed FBs and ICs have a unique identity and provide a unique information hook that needs to cover the description of a complex and eventually distributed collection of FBs. The eventual implementation of the information hook of a composed FB is to be chosen at implementation time and is not part of information flows at the conceptual level. However, we identify two possible implementation approaches:

1. The composed FB exposes and describes its internal structure via the information hook, including a list of the internal FBs that are part of the composed FB and their relationships, i.e., "connections" between FBs. It is then the responsibility of the information consumer to further access the information hooks and the functions of the internal FBs to obtain the information it requires. This situation is illustrated in Figure 6-6.
2. The composed FB does not expose its internal structure, but provides information about it directly via its information hook, respectively through its other functions. In this case, the composed FB may for example use internal monitoring services to collect and present the distributed information about itself in a coherent and efficient way. This situation is illustrated in Figure 6-7.

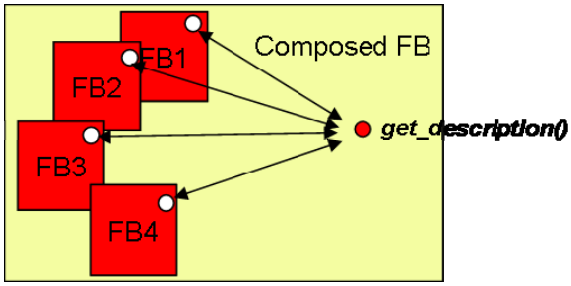



**Figure 6-6: Composed FB exposes its internal structure via the Information Hook**

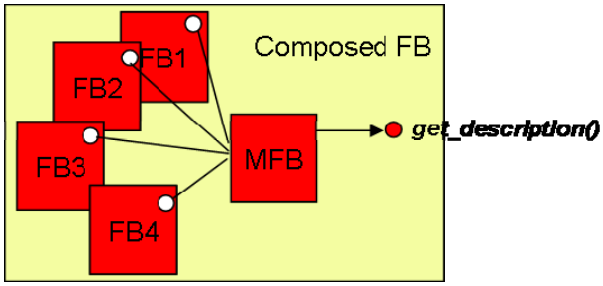



**Figure 6-7: Composed FB uses available monitoring services to build up its service description**

### 6.3.1.4 Interworking FBs

Section 3.4.2 describes inter-compartment communication in ANA. It is achieved in adjacent compartments through the use of Interworking Functional Blocks (IFBs) which are conceptually part of both the adjacent compartments. Thus, IFBs are the entities that are able to "translate" syntax and semantics between the two compartments.

From the information flow concept viewpoint, the necessity to have such IFBs means that they must be able to perform two kinds of operations if necessary: First, they should be able to map entity descriptions from one syntax to another. Second, translating syntax is

not always sufficient to enable understanding between adjacent compartments, but the IFBs may need also to perform semantic mappings.

### 6.3.2 Interactions between ANA entities

As previously stated, *Information Flows* address two practical issues. The first one is related to the description of an ANA entity, by introducing the *Information Hook* concept. The second aspect deals with the actual interaction between ANA entities which compose a generic function chain. We exemplify this second aspect in this section.

To provide clear details about how interactions with a FB take place, let us consider the example depicted in Figure 6-8. The figure shows a particular case in which three input functions of a FB are bound to the upper four IDPs. These three functions are executed when messages/data are received from the corresponding IC/IDP. Two of these functions, called *receive_and_dispatch()*, represent a generic type of input functions. Their role is to receive input data and dispatch it internally (within the FB) to other functions if necessary. This kind of generic input function is for example needed when a single FB can support an arbitrary number of FB instances, like TCP FB could support several independent TCP connections. The received input contains enough information that these functions are able to perform the dispatching, e.g., to execute the function of the particular FB instance. The third input function, i.e. *get_description()*, is part of the information hook and is the function that is executed when another ANA entity requests a description of this FB.

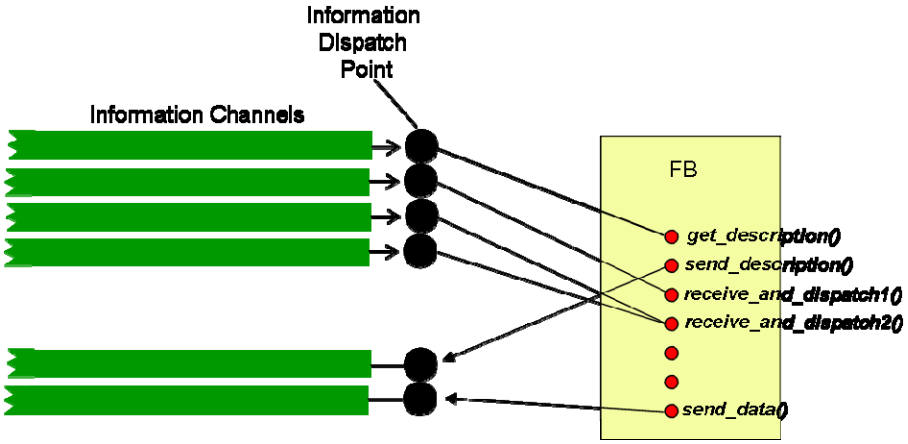



**Figure 6-8 - FB Architectural view for Information Flow**

The figure shows two different generic input and dispatch functions in order to illustrate the fact that a FB is naturally free to choose the way it wishes to be interacted with. For example, one of these *receive_and_dispatch* functions could be handling received data of one instance (running thread) of the FB, such as an end-point of a TCP connection, and the other function handles the received data of another instance. Thus, depending on the situation a FB may wish to perform demultiplexing of incoming data. In our example, we consider two different cases:

(1) *get_description()* function: no (FB internal) *demux* operation has to be performed and the function is directly invoked;

(2) *receive_and_dispatch()* functions: FB internal dispatching is performed.

Outgoing data is sent to the corresponding IDPs via the ANA node internal packet dispatch (see section 3.1.2 of the ANA blueprint).

## 6.3.3 Communication Setup Example

### 6.3.3.1 Overview

We present an example that shows how Information Flows can be used in ANA. We describe the role of information flows in the process of establishing communication between two nodes via e.g., an "Ethernet" Network Compartment. This example describes a rather basic scenario of ANA that is first introduced in section 5.4.5. We explain how information flows support the setup of a communication channel between two Node Compartments (which we call *Node Compartment 1* and *Node Compartment 2*) through an *Ethernet Network Compartment*.

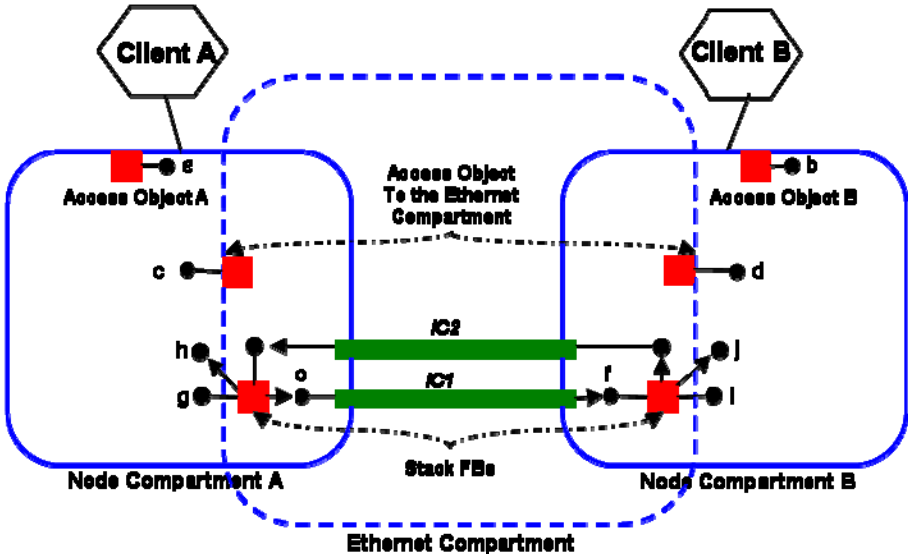



**Figure 6-9 - Communication setup scenario**

Figure 6-9 illustrates the scenario. We use similar notation as in section 3.2.3 and section5.4.5:

- the *Clients A* and *B* represent two applications which have joined the *Node Compartments A* and *B*, respectively;
- the *Access Objects A* and *B* are the FBs responsible for granting the clients access to the corresponding Node Compartment (they handle all control requests sent by the clients to the node compartment abstraction, for further details refer to section 3.2.3);

- the FBs which are bound respectively to the *IDP c* and *IDP d* represent the access objects responsible for granting access to the Network Compartment.
- the *IC1* and *IC2* represent the two information channels which allow communication between the node compartments. Since an IC is a unidirectional communication entity we provide two channels.

For simplicity, we focus in this example only on the Level 2 API. The Level 2 interfaces of the ANA entities shown above are illustrated in Figure 6-10. Note that each of the interfaces includes the two functions that compose the information hook (*get_description()* and *send_description()*) and the entity specific functions.

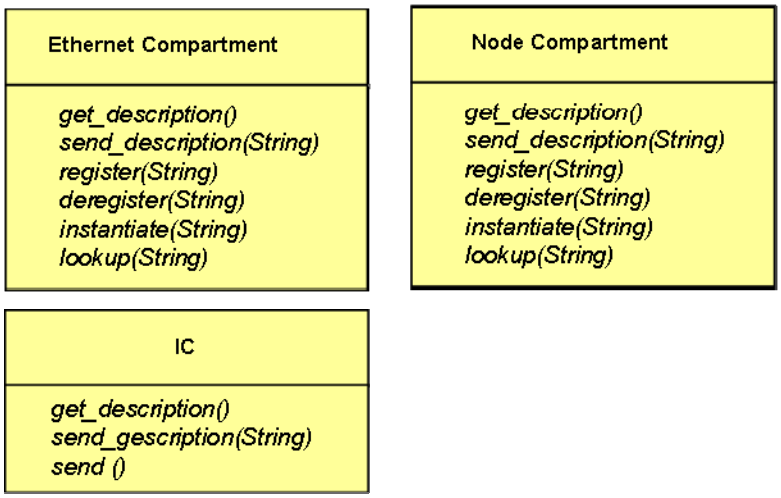



**Figure 6-10: Level 2 Interfaces**

### 6.3.3.2 Stepwise Walkthrough with Sequence Diagram

In Figure 6-11 we show in the form of sequence diagrams, how information flows support the communication channel setup between client A and client B. The numbered steps below correspond to those in the example of section 5.4.5. We only added the interactions with the information hook:

- *Step 1*: Client *A* attaches to the default ANA Node Compartment on its runtime system each ANA client requests access to the functional block in charge of the (private view of the) node compartment;
- *Step 2*: Client *A* receives in return an IDP label *a* for control communication with the ANA node compartment. the requests to access the access object of the node compartment respectively return *IDP a* for *Client A* and *IDP* b for *Client B*;
- *Step 3*: Client *A* registers an IDP label *b* for future result messages and maps it to the treating function *Function1*
- Now possible: optional use of node compartments control FBs information hook. In order to receive a description of the interface of the node compartments control FB, client A can invoke the function *get_description()* at node compartments control FB via IDP a. The node compartment processes the request for information and returns the requested information with function

*send_description()* via IDP b back to client A. The client can simply learn how to instantiate the Ethernet compartment (done at the next step) or learn how to ask for available compartments that could be instantiated.

- *Step 4*: Client *A* then asks the node compartment to instantiate the Ethernet compartment
- *Step 5*: Client *A* receives an IDP label *c* for control communication with the Ethernet compartment
- *Step 6*: Client *A* then registers an IDP label *d* for future control responses of the Ethernet compartment and maps it to the treating function *Function2*
- At this point: optional use of information hook of the access object to Ethernet compartment. Client A can invoke *get_description()* from the access object to the Ethernet compartment via IDP c, and the access object is processing this request and returning the information to client A by calling *send_description()* and using IDP d. At this point the description can tell the client how to lookup other members of the Ethernet compartment and know the load of the Ethernet compartment or how to query it.
- *Step 7*: Client *A* then requests the Ethernet compartment to lookup the client *B*
- *Step 8*: Client *A* receives in return the IDP label *e* to communicate with client *B*
- At this point: optional use of information hook of the IC. The information learned can be how to transmit data via the IC and which is the priority or available bandwidth on this IC or how to query such information via the IC's interface.
- *Step 9*: Client *A* can access/ talk to client B to access the available services at client B.

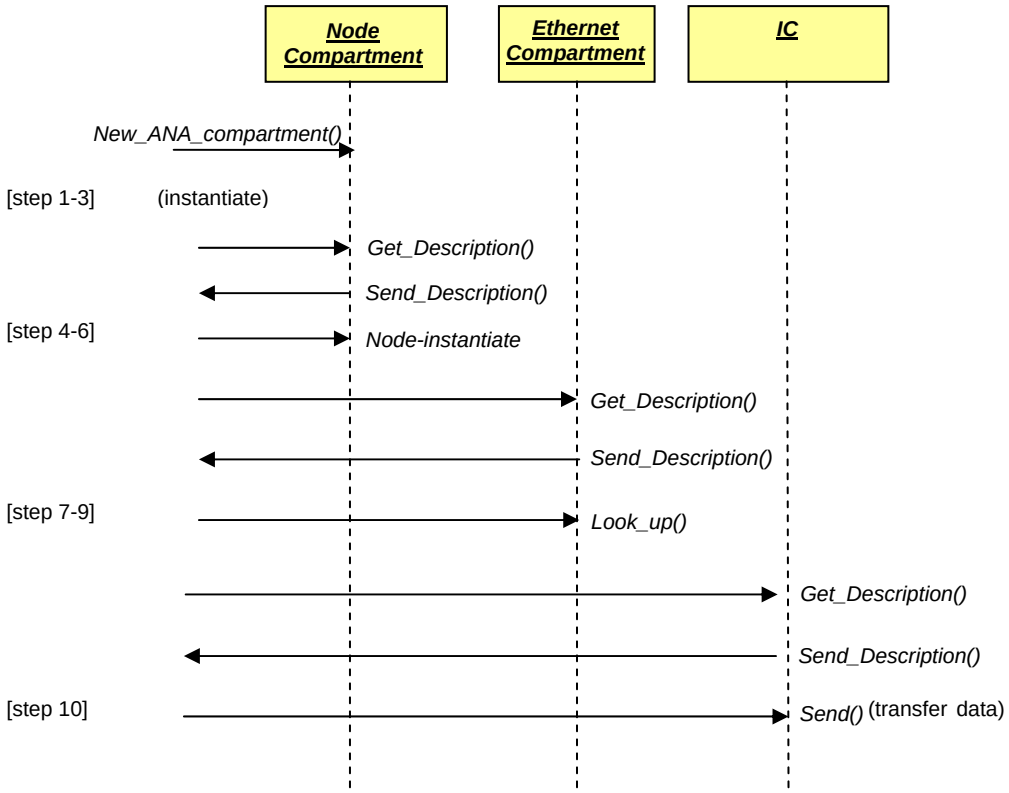



**Figure 6-11 - Information Flows (Level 2 API view)**

### 6.3.3.3 Example descriptions of ANA entities using IDL and RDF/XML

As we have pointed out in earlier sections, the exact format of the description of a particular ANA entity that can be obtained via its information hook is mandated by agreed policies of the given compartment. It is therefore not "hardwired" in the core architecture. However, these entities do not necessarily have any prior information of each other since compartments cannot be considered static. After all, exchanging this information between entities is a task where the entity description obtainable via the information hook has the key role. Thus, the task of providing a self-description that is understandable by other entities in a particular compartment is non-trivial. That is why we discuss in this section potential solutions for this task.

The challenge is composed of enabling two levels of understanding between all ANA entities that comprise a compartment: The entities must understand the syntax and the semantics of a description of a given ANA entity. In other words, a given ANA entity must be able to parse a given description and understand the syntax. In addition, there must be a minimal agreement of a common vocabulary, i.e. the entity parsing the description must be able to understand the meaning of the parsed elements. In order to establish understanding of the syntax of a description, a common description language is needed. On the other hand, ontologies can essentially be considered as dictionaries that enable the establishment of a common understanding of the semantics.

We consider here an example where we describe ANA entities from two perspectives: First, we use IDL (Interface Definition Language) [26], which is well established language used to describe a software component's interface, to describe the interface of an ANA entity. This information enables the entity that interprets the description to be able to interact with the entity, e.g., to perform remote procedure calls via the Level 2 API. Note that this information does not necessarily guarantee that the interpreting entity understands what a particular function does, i.e. its semantics, even though it is able to execute it. It is naturally possible that a standard set of functions/procedures are known by all members of a particular compartment. As a second step, we use RDF (Resource Description Framework) to describe the entity's purpose, the semantics of each function of the interface etc. Our RDF examples are encoded with XML [25]. This second type of description tells the interpreting entity what exactly the other entity can do. For simplicity we describe only one FB and one IC. Note that none of the examples is aimed to exemplify the full potential of the approaches but rather to show possible approaches to tackle the problem. A full fleshed RDF/XML example of something as rich as the ANA world can be exceedingly complex.

The IDL description of the Level 2 objects are shown below.

- Node Compartment

```
interface anaCompartment
{
  string getDescription();
  string register(String id);
  string deregister(String id);
  string instantiate(String id);
  string lookup(String id);
};
```

- "Ethernet" Compartment:

```
interface anaCompartment
{
  string getDescription();
  string register(String id);
  string deregister(String id);
  string instantiate(String id);
  string lookup(String id);
};
```

- IC

```
interface anaCompartment
{
  string getDescription();
  int send();
};
```

For the RDF examples, we define a very simple ontology, *adl (ANA Description Language)* in RDF in a similar fashion than the authors defined *ndl* in [23]. In RDF, information is described in triplets that consist of *subject (the class being described), property (the property the statement describes), and object (the value of the property according to the statement).* We define the following 3 classes and 10 properties in this example adl:

| **Classes** | **Properties** |
|---|---|
| FB | Id |
| IC | belongsTo |
| Compartment | priority |
| | bandwidth |
| | capacity |
| | Find |
| | Start |
| | Registration |
| | Deregistration |
| | Transmit |

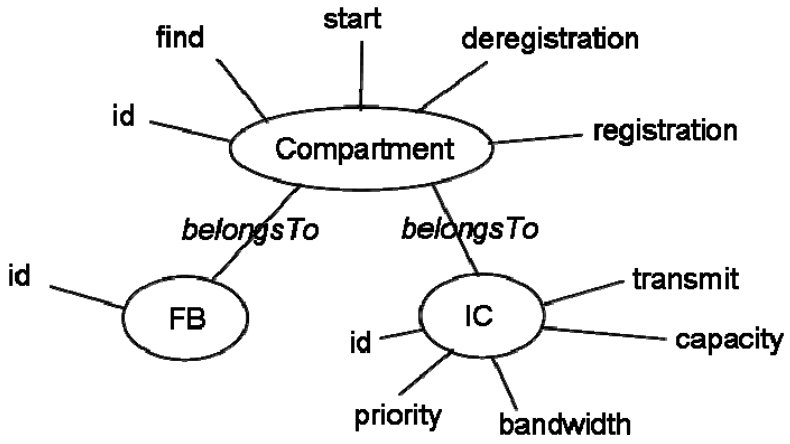



**Figure 6-12: Classes and properties in adl.**

Figure 6-12 illustrates the classes and properties that we define in adl. The id property attaches an identifier to a FB, an IC, or a compartment. *BelongsTo* property describes the relation of IC or FB being part of a compartment. Priority, bandwidth, and capacity are properties of an IC. By using the above resources and properties, we obtain the following descriptions of the objects in the Level 2 API example.

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"

<!—Ethernet compartment-->
<adl:Compartment rdf:about="#AnaEth">
<adl:id>AnaEth</adl:id>
<adl:registration>register</adl:registration>
<adl:deregistration>deregister</adl:deregistration>
<adl:start>instantiate</adl:start>
<adl:find>lookup</adl:find>
</adl:Compartment>

<!—Node compartment-->
<adl:Compartment rdf:about="#NodeCmp">
<adl:id>NodeCmp</adl:id>
<adl:registration>register</adl:registration>
<adl:deregistration>deregister</adl:deregistration>
<adl:start>instantiate</adl:start>
<adl:find>lookup</adl:find>
</adl:Compartment>

<!--IC-->
<adl:IC rdf:about="#IcA">
<adl:id>IcA</adl:id>
<adl:capacity>100Mbit/s<adl:capacity>
<adl:bandwidth>50Mbit/s<adl:bandwidth>
<adl:priority>2</adl:priority>
<adl:transmit>send<adl:transmit>
   <adl:belongsTo rdf:resource="#AnaEth"/>
</adl:Interface>

</rdf:RDF>
```

Note that in the above we describe also the functions of the interface even if in a trivial way, e.g., *start* property's object is *instantiate* which refers to the name of the function in the IDL description. Thus, we assume in the above example that the semantics of the standard functions are not known but they become clear only after the interpretation of the adl language description. For example, the property *start* in adl language has a particular semantic meaning and in that way explains what the function *instantiate*() does. Describing the interface functions in this way also allows the interface of a particular ANA entity to be defined in any way. For instance, a compartment's interface could contain a function *join()* and another compartment's interface a function *instantiate()* both with the same semantics which can be described in the adl language with the property *start.*

In addition to describing the interface functions, we also included other properties into the above example. For example, the IC description contains *capacity, bandwidth,* and *priority,* which can be seen as configuration and monitoring information. We have added directly values for these properties but another approach would be to instead describe those interface functions that provide the particular information. For instance, one could

imagine having a function called *QoS()* which accepts a string argument that can have a value "priority".

### 6.3.4 Supported Information Sharing Methods

As described in the requirements analysis for the ANA architecture, ANA should support different information sharing methods. The following two methods are identified as necessary information sharing methods in ANA:

- It should be possible to push information from an information provider to an information consumer (push model).
- It should be possible that information consumers pull information from an information provider (pull model).

Pull type of interaction happens as described in Figure 6-4 and Figure 6-5. In other words, first an entity description is requested using *get_description()* message and then returned using *send_description()*. Afterwards, in the second phase, the requesting party uses the interface of the information providing party to "pull" the information in request/reply fashion. Push model is different in the way that the description is directly sent using *send_description()* with no prior *get_description()*. If the corresponding FB supports the push model, it will accept the entity description which describes the information it is about to receive. Then, in the second phase, the information itself is pushed. The second phase implies that the receiving FB that supports the push model interprets the description and prepares, if necessary, a corresponding input to accept the information.

We identified also in the requirements analysis the need for supporting different communication paradigms. For instance, it should be possible to address a group of consumers in multicast style. Information channel is the abstraction within ANA that provides means to communicate. As specified in the section 3.1.2, it captures various types of communication channels, such as *multi-cast*, *any-cast*, *uni-cast*, or *con-cast*. Thus, when using push model, for instance, a FB can broadcast or multicast its description to a group of recipients via a corresponding IC, for example. This applies to both phases, i.e. sending the description and the information itself.

We also identified in the requirements analysis the necessity to prioritize certain information flows because some information is more time critical than other. We also note here that such features are essentially features that can be provided by information channel. These properties can be nevertheless accessed and potentially also modified through the interface of the IC which can be obtained through its information hook.

# 7 CONCLUSIONS

As stated previously in this document, the main objective of the Blueprint is to describe the reference model of the ANA operation. The purpose of this design effort is to identify and describe the basic abstractions and paradigms of ANA and their interactions and impact in and on the overall architecture. A key objective is also to guide the prototyping of the ANA communication system throughout the project.

At the coarsest level, the core abstraction of ANA is the "Compartment" which encompasses the notions of networks and realms and is the basic unit for the interaction and federation of networks. A key feature is that compartments reduce complexity by hiding the compartment internal communication complexity to the "outside world", which interfaces with a given compartment via the communication abstractions provided by the compartment. Hence networks can be decomposed into smaller and easier manageable units: each compartment has indeed full autonomy on how to handle internal communication (i.e. naming, addressing, routing, etc). The boundaries of a compartment are typically based on technological and/or administrative boundaries or policy domains. ANA anticipates that many compartments co-exist and will provide the necessary functionalities such that compartments are able to interwork on various levels.

At a more detailed level, compartments contain three kinds of abstractions: Functional Blocks (FBs), Information Channels (ICs), and Information Dispatch Points (IDPs). These abstractions are used to model the "internals" of a compartment and the service the compartment offers. At this granularity level, an important component is the information flow framework, which defines a generic method to obtain detailed information about the capabilities and properties of FBs and ICs. The degree of details at which a compartment exposes its internal abstractions is specific to each compartment and ranges from full visibility to total opaqueness.

At the lowest level, the underpinning component is the ANA Node, which is the entity that hosts the ANA elements and functionalities. It is the main interface to the host operating system or underlying hardware, and also provides an adaptation layer allowing legacy applications to be reused in an ANA context. The core and mandatory part of the ANA Node is the MINMEX framework, which provides the basic low-level functionalities required to bootstrap and run an ANA node. Beside the MINMEX framework, the ANA node provides a development framework – the "Playground" – where the more elaborated and complex networking functionalities of ANA are placed. For example, compartment-specific modules are executed in the Playground.

All together, these concepts and abstractions form the first version of the ANA Blueprint. This reference model will be used to develop the first version of the ANA prototype, which, along with a testbed infrastructure, will be used to validate the core ANA functionalities and demonstrate autonomicity (e.g., self-configuration and self-organisation) on a compartment basis.

The initial feedback obtained during this first prototyping phase (M13-M24) has been used to extend and refine the version 1.0 of the Blueprint. Further feedback that will be gained during the remaining prototyping phase will eventually lead to the specification of a mature version of the Blueprint sometime around December 2008. This will mark the start of the final integration phase where advanced autonomic functionalities will be demonstrated.

# 8 REFERENCES


[1] S. Schmid, M. Sifalakis and D. Hutchison, Towards Autonomic Networks. In Proc. of 1st International IFIP TC6 Conference on Autonomic Networking (AN 2006), Lecture Notes in Computer Science 4195, Paris, France, September 2006, pp. 1-11.

[2] EU FP6 FET Project Proposal on Autonomic Network Architecture (ANA), FP6-IST-27489, March 2005.

[3] D1.7: Prototype implementation of the information flow framework. Workpackage 1 Deliverable 1.7, ANA Project FP6-IST-27489, December 2007.

[4] D1.8: Prototype implementation of the core ANA software. Workpackage 1 Deliverable 1.8, ANA Project FP6-IST-27489, December 2007.

[5] D2.1: First Draft of Routing Design and Service Discovery. Workpackage 2 Deliverable 2.1, ANA Project FP6-IST-27489, December 2006.

[6] D2.2: Functional Composition Framework. Workpackage 2 Deliverable 2.2, ANA Project FP6-IST-27489, December 2006.

[7] D2.3: Initial Design and Evaluation of Inter-Compartment Communication Schemes. Workpackage 2 Deliverable 2.3, ANA Project FP6-IST-27489, December 2007.

[8] D2.4: Initial Design and Prototype Implementation of the Functional. Workpackage 2 Deliverable 2.4, ANA Project FP6-IST-27489, December 2007.

[9] D2.6: Design and Evaluation of Self-Association and Self-Organization mechanisms for Network Compartments. Workpackage 2 Deliverable 2.6, ANA Project FP6-IST-27489, December 2007.

[10] D2.7: Design, Prototype and Evaluation of an Customizable Overlay Compartment. Workpackage 2 Deliverable 2.7, ANA Project FP6-IST-27489, December 2007.

[11] D2.8 Service Discovery and Routing Schemes for intra- and inter-Compartment Service Provisioning. Workpackage 2 Deliverable 2.8, ANA Project FP6-IST-27489, December 2007.

[12] D3.1: Monitoring Framework. Work Package 3 Deliverable 3.1, ANA Project FP6-IST-27489, December 2006.

[13] D3.2: Resilience/Security Framework. Workpackage 3 Deliverable 3.2, ANA Project FP6-IST-27489, December 2006.

[14] D3.3: The Monitoring Part of the ANA Architecture (v1) . Workpackage 3 Deliverable 3.3, ANA Project FP6-IST-27489, December 2007.

[15] D3.4: The Self-Optimization Part of the ANA Architecture (v1) . Workpackage 3 Deliverable 3.4, ANA Project FP6-IST-27489, December 2007.

[16] D3.5: Specification of failure detection and fault management in the ANA architecture. Workpackage 3 Deliverable 3.5, ANA Project FP6-IST-27489, December 2007.

[17] D3.6: The Resilience Part of the ANA Architecture (v1). Workpackage 3 Deliverable 3.6, ANA Project FP6-IST-27489, December 2007.

[18] D. Clark, J. Wroclawski, K. R. Sollins and R. Braden. Tussle in Cyber-space: Defining Tomorrow's Internet. In Proc. of ACM SIGCOMM, Pittsburgh, PA, USA, August 19-23, 2002, pp. 347-356.

[19] J. Pujol, S. Schmid, L. Eggert, M. Brunner and J. Quittek. Scalability Analysis of the TurfNet Naming and Routing Architecture. In Proc. of 1st ACM Workshop on Dynamic Interconnection of Networks (DIN 2005), Cologne, Germany, September 2, 2005, pp. 28-32.

[20] D. Clark, R. Braden, A. Falk and V. Pingali. FARA: Reorganizing the Addressing Architecture. In Proc. of ACM SIGCOMM Workshop on Future Directions in Network Architecture (FDNA), Karlsruhe, Germany, Au-gust 2003, pp. 313-321.

[21] D1.3: Information Flow Requirements. Workpackage 1 Deliverable 1.3, ANA Project FP6-IST-27489, December 2006.

[22] D1.2: ANA Requirements. Workpackage 1 Deliverable 1.2, ANA Project FP6-IST-27489, December 2006.

[23] Resource Description Framework (RDF), URL http://www.w3.org/RDF/

[24] Jeroen van der Ham, Freek Dijkstra, Franco Travostino, Hubertus Andree, and Cees de Laat. "Using RDF to Describe Networks". In Future Generation Computer Systems, Feature topic iGrid 2005. 2006.

[25] D. Beckett and B. McBride. RDF/XML Syntax Specification. February 2004. http://www.w3.org/TR/rdf-syntax-grammar/

[26] Specification of OMG IDL. http://www.omg.org/cgi-bin/doc?formal/02-06-39.