Project Number:	**FP7-257123**
Project Title:	**CONVERGENCE**
Deliverable Type:	**Report**
Dissemination Level	**Public**

Deliverable Number:	**D3.2**
Contractual Date of Delivery to the CEC:	**31.05.2011**
Actual Date of Delivery to the CEC:	**30.07.2011**
Title of Deliverable:	**System architecture**
Workpackage contributing to the Deliverable:	**3**
Nature of the Deliverable:	**Report**
Editors:	**Mihai Tanase, Lucian Corlan and Stefano Salsano**
Authors:	**Stefano Salsano, Andrea Detti, Giuseppe Tropea, Nicola Blefari Melazzi (CNIT), Leonardo Chiariglione (CEDEO), Helder Castro (INESC), Angelos – Christos Anadiotis, Aziz Mousas, Charalampos Patrikakis (ICCS), Thomas Huebner (Morpho), Mihai Tanase, Lucian Corlan (UTI), Panagiotis Gkonis (SIL), Jose Ribas, Daniel Sequeira (WIPRO)**

Keyword List:	**Architecture, components, services, applications, middleware, network, API, requirements, cross-concerns.**

Executive Summary

This document supersedes the previous deliverable of WP3 (D3.1) and presents a comprehensive description of the CONVERGENCE architecture and the technical components of the framework, with an emphasis on the technical innovations that this project is proposing. The document is organized in three main sections, which **progressively** detail the technical achievements of the CONVERGENCE system. These sections are:

- Section 3 presents an overview of the CONVERGENCE system, which provides a revised high-level architecture of the CONVERGENCE system and introduces the main CONVERGENCE concepts.
- Section 4 presents a detailed description of the main technical and architectural concepts. This section is organized as a set of monographs. Each monograph provides a self contained description of a feature of CONVERGENCE at the highest level of abstraction at which the reader can understand which requirements have been addressed, and the main characteristics of the solutions envisaged.
- Sections 5, 6, and 7 further detail the technical specification of the architectural components. The focus is on the technical elements at the Middleware level (section 6), and at the Computing Platform level (section 7). Section 5 very briefly describes the Application level and gives only an overview of the middleware interactions with tools and application, as the main elements of the Application level are described in other deliverables (e.g. D7.1).

Additionally, the APIs for the Technology Engines of the CoMid are reported in ANNEX A, providing the maximum level of detail about the specification of the Middleware to the interested reader.

Thanks to this organization, depending on her interest, the reader can stop after reading section 3, can read some parts of section 4, can stop after reading all monographs of Section 4, or can proceed further in the detailed technical specification sections and even delve into the APIs definition. However, we point out that the bulk of technical specification is contained in sections 6 and 7.

An important design requirement and characteristic of CONVERGENCE is modularity, a feature built on standard interfaces. CONVERGENCE is not a monolithic system: it is designed so that Applications, Middleware, Network and Security protocols and mechanisms can work independently from each other. This has obvious advantages: if we find that Content-Centric Networking (CCN) is not a viable solution, we can use plain IP networking without having to sacrifice CONVERGENCE Middleware and Applications. Likewise, the CONVERGENCE CCN-based networking approach can be deployed as a new networking paradigm, independently from the CONVERGENCE Middleware and Applications. Modularity is applied also *within* levels: for instance the middleware could use a different approach to support some publish/subscribe operations, to the CDS and the semantic overlay. Modularity is also important for take up of CONVERGENCE. The fact that users can adopt "market-ready" parts of CONVERGENCE without adopting the whole system facilitates migration from current systems.

However, achieving such a level of modularity, hand-in-hand with coordinated operation at all system levels, in a distributed environment, has been and is a major technological challenge for the project, which can be met only with a knowledgeable and well-thought design of architecture and interfaces.

Executive Summary 2

1 Introduction 11

2 Terms and Definitions 13

3 Overview of CONVERGENCE system 20

3.1 The CONVERGENCE Architecture 21

3.1.1 Applications 24

3.1.2 Middleware 24

3.1.3 Computing Platform 26

3.1.4 A distributed view of the CONVERGENCE system 26

3.2 CoMid features 27

3.2.1 The VDI 27

3.2.2 Semantic and Dictionaries 28

3.2.3 Publish/Subscribe 29

3.3 CoNet features 30

3.4 CONVERGENCE Security and CoSec 31

3.4.1 Overview 31

3.4.2 CoMid Security 32

3.4.3 The CoSec 33

3.4.4 CoNet security 34

3.4.5 Cryptographic primitives and protocols in CoSec 34

4 Main technical and architectural concepts 36

4.1 The MPEG-M standard 36

4.2 Content Centric Networking 38

4.3 VDI 40
4.4 VDI dynamics and linking 42
4.4.1 Functional Requirements 42
4.5 CDS 43
4.5.1 CDS supports content description 43
4.5.2 CDS dictionaries 45
4.5.3 Mechanisms for exploiting CDS dictionaries 47
4.6 Semantic Overlay 49
4.6.1 Semantic Foundations 49
4.6.2 Semantically managing the Overlay Topology 50
4.6.3 Propagation Protocol 51
4.6.4 Peers Registration to Fractals 51
4.6.5 Message Propagation in Semantic Overlay 53
4.7 Event reporting 55
4.8 Rights Expression Language 55
4.9 Content identification 56
4.10 Publish / subscribe pattern 58
4.10.1 Description 58
4.10.2 Part 1: inserting/storing a semantic subscription system wide 60
4.10.3 Part 2: matching a subscription 61
4.10.4 Part 3: delivering matches to subscribers 61
4.11 Digital Forgetting 61
4.11.1 Introduction 61
4.11.2 Functional Requirements 62

4.11.3 Digital Forgetting Mechanisms 63
4.12 Real world descriptors 66
4.13 Semantic Search and Content Matching 66
4.14 CoMid Security 73
4.14.1 Roles 73
4.14.2 Assets 74
4.14.3 Threats 76
4.14.4 Objectives 77
4.14.5 Security Functional Requirements 79
4.15 CoNet Security 83
4.15.1 CoNet Service Integrity 84
4.15.2 CoNet Producer/User Privacy 86
4.16 Scalability 86
4.16.1 CoMid scalability 87
4.16.2 CoNet scalability 88
5 Overview of Application Level 90
6 Technical specification of CONVERGENCE Middleware (CoMid) 92
6.1 CoMid overview 92
6.1.1 CoMid functionality 92
6.1.2 Elementary Services, Protocol Engines and Technology Engines 93
6.2 Orchestration and Aggregation 93
6.3 Protocol engines 95
6.4 CDS TE 97
6.4.1 General description 97

6.4.2 Main components and their interaction........97
6.5 CoNet TE........99
6.5.1 Main components and their interaction........99
6.6 Digital Item TE........99
6.6.1 Main Components and their Interaction........100
6.7 Event Report TE........100
6.7.1 Event Report Request (ER-R) message........101
6.7.2 Event Report (ER) message........101
6.7.3 Main components and their interaction........102
6.8 Media Framework TE........104
6.8.1 Main components and their interaction........104
6.9 Match TE........104
6.9.1 General description........104
6.9.2 Main components and their interaction........105
6.10 Metadata TE........106
6.10.1 Main Components and their Interaction........106
6.11 MPEG-21 File TE........107
6.11.1 Main Components........108
6.11.2 MPEG-21 File Format........108
6.12 Overlay TE........108
6.12.1 Main components and their interaction........108
6.13 REL TE........111
6.13.1 Main components and their interaction........112
6.14 Security TE........113

6.14.1 Main components and their interaction 113
7 Technical specification of Computing Platform level 114
7.1 Network component 114
7.1.1 CoNet Architecture 116
7.1.2 Model of operations 119
7.1.3 CoNet protocol stack 120
7.1.4 Name-based routing: lookup-and-cache 122
7.1.5 Integrating CoNet in IP 123
7.1.6 CoNet Application Program Interface 124
7.1.7 Examples of use of the CoNet Application Program Interface 127
7.2 Security Component (CoSec) 131
7.2.1 Survey 131
7.2.2 Cryptographic Primitives 132
7.2.3 Smart Cards in CONVERGENCE 135
7.2.4 Cryptographic Protocols for specific Security Functional Requirements 135
7.2.5 Authentication of users 137
7.2.6 Integrity/Authenticity 137
7.2.7 Licensing 137
7.2.8 Example Protocols 137
8 Bibliography 139
9 ANNEX A - APIs for CoMid Technology Engines 142
9.1 CDS TE API 142
9.2 CoNet TE API 144
9.3 Digital Item TE 145

9.4 Event Report TE API 148
9.5 Media Framework TE API 150
9.6 Match TE API 150
9.7 Metadata TE API 151
9.8 MPEG-21 File TE API 155
9.9 Overlay TE API 157
9.9.1 MessageCreator 157
9.9.2 MessageKeeper 157
9.9.3 MessageParser 157
9.9.4 PropagationMessageHandler 157
9.9.5 RegistryCreator 158
9.9.6 RegistryParser 158
9.10 REL TE API 159
9.10.1 Rights expression creation 159
9.10.2 Rights expression parsing 160
9.10.3 Authorization Manager 162
9.10.4 Condition Manager 162
9.11 Security TE API 162
9.11.1 CertificateManager Interface 163
9.11.2 KeyManagerInterface 164
9.11.3 SecureDeviceManagerInterface 167
9.11.4 SecureRepositoryManagerInterface 168
9.11.5 SecurityEngine Class 170
9.11.6 SecurityEngineKeys Interface 170

9.11.7 SecurityInfoCreatorInterface 172

9.11.8 SecurityInfoParserInterface 172

10 ANNEX B – Survey of solutions of real world descriptors 174

1 Introduction

This document describes the technical concepts and the functional architecture for the CONVERGENCE framework, based on the use cases and functional requirements defined in the Work Package 2 (WP2). The purpose of the CONVERGENCE system is to propose a novel content-centric framework that complements and enhances the current Internet architecture. The Convergence framework is built upon the MPEG-M (MXM) standard and therefore uses and expands on its concepts.

The presentation of the CONVERGENCE architecture is organized by providing four different views with increasing technical detail:

- Overview of CONVERGENCE system (section 3). This section provides the Convergence system architecture overview by:
  - Describing the high-level architecture diagrams of the Convergence system
  - Defining the main technical and architectural concepts of the system
- Description of the main technical and architectural concepts (section 4). This section is organized as a set of monographs. Each monograph provides a self contained description of a feature of CONVERGENCE at the highest possible level of abstraction that lets the reader understand which requirements have been addressed, and the main characteristics of the envisaged solutions.
- Technical specification of the architectural components: it comprises sections 5, 6, and 7 providing the detailed definition of all the modules at the different levels, in particular the Middleware level (section 6) and the Computing Platform level (section 7). The Application level (section 5) is only sketched, as it is covered in the deliverables of other WPs (e.g. D7.1). In order to ease the reading, the specification of the APIs for the different components of the Middleware level is reported separately.
- APIs definition for Technology Engines of the CoMid, reported in an ANNEX.

Thanks to this organization, depending on her interest, the reader can stop after reading section 3, can read some parts of section 4, can stop after reading all monographs of Section 4, or can proceed further in the detailed technical specification sections and even delve into the APIs definition. However, we point out that the bulk of technical specification is contained in sections 6 and 7.

The main results achieved by this deliverable are:

- A 3-levels architecture (Computing Platform, Middleware, Applications) of an ICT system
- Specification of the Middleware (CoMid), inspired by the middleware standardized by MPEG
- Specification of a content-centric network (CoNet) and its integration in the architecture
- Specification of security technology (CoSec) and its integration in the architecture
- Specification of APIs between the modular system components
- Extensions of the VDI format to support semantic linkage and integration in the architecture. The proposed format extends the DI format standardized by MPEG.
- Specification of a Publish/Subscribe mechanism and its integration in the architecture
- Specification of a Digital Forgetting mechanism and its integration in the architecture
- Specification of an ontology service and its integration in the architecture.

An important design requirement and characteristic of CONVERGENCE is modularity, a feature built on standard interfaces. CONVERGENCE is not a monolithic system: it is designed so that Applications, Middleware, Network and Security protocols and mechanisms can work independently from each other. This has obvious advantages: if we find that Content-Centric Networking (CCN) is not a viable solution, we can use plain IP networking without having to sacrifice CONVERGENCE Middleware and Applications. Likewise, the CONVERGENCE CCN-based networking approach can be deployed as a new networking paradigm, independently from the CONVERGENCE Middleware and Applications. Modularity is applied also *within* levels: for instance the middleware could use a different approach to support some publish/subscribe operations, to the CDS and the semantic overlay. Modularity is also important for take up of CONVERGENCE. The fact that users can adopt "market-ready" parts of CONVERGENCE without adopting the whole system facilitates migration from current systems.
However, achieving such a level of modularity, hand-in-hand with coordinated operation at all system levels, in a distributed environment, has been and is a major technological challenge for the project, which can be met only with a knowledgeable and well-thought design of architecture and interfaces.

2 Terms and Definitions

Term	Definition
Access Rights	Criteria defining who can access a VDI or its components under what conditions.
Advertise	Procedure used by a CoNet user to make a resource accessible to other CoNet users.
Application	Software, designed for a specific purpose that exploits the capabilities of the CONVERGENCE System.
Business Scenario	A scenario describing a way in which the CONVERGENCE System may be used by specific users in a specific context or, more narrowly, a scenario describing the products and services bought and sold, the actors concerned and, possibly, the associated flows of revenue in such a context.
Clean-slate architecture	The CONVERGENCE implementation of the Network Level, totally replacing existing IP functionality. See “Integration Architecture” and ””Overlay Architecture” and “Parallel Architecture”.
CoApp	The CONVERGENCE Application Level.
CoApp Provider	A user providing Applications running on the CONVERGENCE Middleware Level (CoMid).
CoMid	The CONVERGENCE Middleware Level.
CoMid Provider	A user providing access to a single or an aggregation of CoMid services.
CoMid Resource	A virtual or physical object or service referenced by a VDI, e.g. media, Real World Objects, persons, internet services. It has the same meaning of “Resource” and it is used only to better specify the term “Resource” when there is a risk of a misunderstanding with the term “CoNet Resource”.
Community Dictionary	A CoMid Technology Engine that provides all the matching concepts in a

Service (CDS)	user's subscription, search request and publication.
CoNet Provider	A user providing access to CoNet services, i.e. the equivalent of an Internet Service Provider.
CoNet Resource	A resource of the CoNet that can be identified by means of a name; resources may be either Named-data or a Named service access point.
Content-based resource discovery	A user request for resources, either through a subscription or a search request to the CONVERGENCE system (from literature). See "subscription" and "search".
Content-based Subscription	A subscription based on a specification of user's preferences or interests, (rather than a specific event or topic). The subscription is based on the actual content, which is not classified according to some predefined external criterion (e.g., topic name), but according to the properties of the content itself. See "Subscription" and "Publish-subscribe model".
Content-centric	A network paradigm in which the network directly provides users with content, and is aware of the content it transports, (unlike networks that limit themselves to providing communication channels between hosts).
CONVERGENCE Applications level (CoApp)	The level of the CONVERGENCE architecture that establishes the interaction with CONVERGENCE users. The Applications Level interacts with the other CONVERGENCE levels on behalf of the user.
CONVERGENCE Computing Platform level (CoComp)	The Computing Platform level provides content-centric networking (CoNet), secure handling (CoSec) of resources within CONVERGENCE and computing resources of peers and nodes.
CONVERGENCE Core Ontology (CCO)	A semantic representation of the CoReST taxonomy. See "CONVERGENCE Resource Semantic Type (CoReST)"
CONVERGENCE Device	A combination of hardware and software or a software instance that allows a user to access Convergence functionalities
CONVERGENCE Engine	A collection of technologies assembled to deliver specific functionality and made available to Applications and to other Engines via an API

CONVERGENCE Middleware level (CoMid)	The level of the CONVERGENCE architecture that provides the means to handle VDIs and their components.
CONVERGENCE Network (CoNet)	The Content Centric component of the CONVERGENCE Computing Platform level. The CoNet provides access to named-resources on a public or private network infrastructure.
CONVERGENCE node	A CONVERGENCE device that implements CoNet functionality and/or CoSec functionality.
CONVERGENCE peer	A CONVERGENCE device that implements CoApp, CoMid, and CoComp (CoNet and CoSec) functionality.
CONVERGENCE Resource Semantic Type (CoReST)	A list of concepts or terms that makes it possible to categorize a resource, establishing a connection with the resource's semantic metadata.
CONVERGENCE Security element (CoSec)	A component of the CONVERGENCE Computing Platform level implementing basic security functionality such as storage of private keys, basic cryptography, etc.
CONVERGENCE System	A system consisting of a set of interconnected devices - peers and nodes - connected to each other built by using the technologies specified or adopted by the CONVERGENCE specification. See "Node" and "Peer".
Digital forgetting	A CONVERGENCE system functionality ensuring that VDIs do not remain accessible for indefinite periods of time, when this is not the intention of the user.
Digital Item (DI)	A structured digital object with a standard representation, identification and metadata. A DI consists of resource, resource and context related metadata, and structure. The structure is given by a Digital Item Declaration (DID) that links resource and metadata.
Domain ontology	An ontology, dedicated to a specific domain of knowledge or application, e.g. the W3C Time Ontology and the GeoNames ontology.
Elementary Service (ES)	The most basic service functionality offered by the CoMid.
Entity	An object, e.g. VDIs, resources, devices, events, group, licenses/contracts, services and users, that an Elementary Service can act upon or with which it can interact.

Expiry date	The last date on which a VDI is accessible by a user of the CONVERGENCE System.
Fractal	A semantically defined virtual cluster of CONVERGENCE peers.
Identifier	A unique signifier assigned to a VDI or components of a VDI.
Integration Architecture	An implementation of CoNet designed to integrate CoNet functionality in the IP protocol by means of a novel IPv4 option or by means of an IPv6 extension header, making IP content-aware. See "Clean-state Architecture", "Overlay Architecture", "Parallel Architecture"
License	A machine-readable expression of Operations that may be executed by a Principal.
Local named resource	A named-resource made available to CONVERGENCE users through a local device, permanently connected to the network. Users have two options to make named-resources available to other users: 1) store the resource in a device, with a permanent connection to the network; 2) use a hosting service. In the event she chooses the former option, the resource is referred to as a local named-resource.
Metadata	Data describing a resource, including but not limited to provenance, classification, expiry date etc.
MPEG eXtensible Middleware (MXM)	A standard Middleware specifying a set of Application Programming Interfaces (APIs) so that MXM Applications executing on an MXM Device can access the standard multimedia technologies contained in the Middleware as MXM Engines.
MPEG-M	An emerging ISO/IEC standard that includes the previous MXM standard.
Multi-homing	In the context of IP networks, the configuration of multiple network interfaces or IP addresses on a single computer.
Named-data	A named-resource consisting of data.
Named resource	A CoNet resource that can be identified by means of a name. Named-resources may be either data (in the following referred to as "named-data")

	or service-access-points ("named-service-access-points").
Named service access point	A kind of named-resource, consisting of a service access point identified by a name. A named-service-access-point is a network endpoint identified by its name rather than by the Internet port numbering mechanism.
Network Identifier (NID)	An identifier identifying a named resource in the CONVERGENCE Network. If the named resource is a VDI or an indentified VDI component, its NID may be derived from the Identifier (see "Identifier").
Overlay architecture	An implementation of CoNet as an overlay over IP. See "Clean-state Architecture" and "Integration Architecture" and "Parallel Architecture"
Parallel architecture	An implementation of CoNet as a new networking layer that can be used in parallel to IP. See "Clean-state Architecture" and "Integration Architecture" and ""Overlay Architecture"
Policy routing	In the context of IP networks, a collection of tools for forwarding and routing data packets based on policies defined by network administrators.
Principal (Rights Expression Language)	The User to whom Permissions are Granted in a License.
Principal (CoNet)	The user who is granted the right to use a *CoNet Principal Identifier* for naming its named resources. For example, the principal could be the provider of a service, the publisher or the author of a book, the controller of a traffic lights infrastructure, or, in general, the publisher of a VDI. A Principal may have several Principal Identifiers in the CoNet.
Principal Identifier (CoNet)	The Principal identifier is a string that is used in the Network Identifiers (NID) of a CoNet resource, when the NID has the form: NID = <namespace ID, hash (Principal Identifier), hash (Label)> In this approach, hash (Principal Identifier) must be unique in the namespace ID, and Label is a string chosen by the principal in such a way

	that hash(Label) is unique for in the context of the Principal Identifier.
Publish	The act of informing an identified subset of users of the CONVERGENCE System that a VDI is available.
Publisher	A user of CONVERGENCE who performs the act of publishing.
Publish-subscribe model	CONVERGENCE uses a content-based approach for the publish-subscribe model, in which notifications about VDIs are delivered to a subscriber only if the metadata / content of those VDIs match constraints defined by the subscriber in his Subscription VDI.
Real World Object	A physical object that may be referenced by a VDI.
Resource	A virtual or physical object or service referenced by a VDI, e.g. media, Real World Objects, persons, internet services.
Scope (in the context of routing)	In the context of advertising and routing, the geographical or administrative domain on which a network function operates (e.g. a well defined section of the network - a campus, a shopping mall, an airport -, or to a subset of nodes that receives advertisements from a service provider).
Search	The act through which a user requests a list of VDIs meeting a set of search criteria (e.g. specific key value pairs in the metadata, key words, free text etc.).
Service Level Agreement (SLA)	An agreement between a service provider and another user or another service provider of CONVERGENCE to provide the latter with a service whose quality matches parameters defined in the agreement.
Subscribe	The act whereby a user requests notification every time another user publishes or updates a VDI that satisfies the subscription criteria defined by the former user (key value pairs in the metadata, free text, key words etc.).
Subscriber	A user of CONVERGENCE who performs the act of subscribing.
Timestamp	A machine-readable representation of a date and time.
Tool	Software providing a specific functionality that can be re-used in several applications.
Trials	Organized tests of the CONVERGENCE System in specific business scenarios.

Un-named-data	A data resource with no NID.
User	Any person or legal entity in a Value-Chain connecting (and including) Creator and End-User possibly via other Users.
User (in OSI sense)	In a layered architecture, the term is used to identify an entity exploiting the service provided by a layer (e.g. CoNet user).
User ontology	An ontology created by CONVERGENCE users when publishing or subscribing to a VDI.
User Profile	A description of the attributes and credentials of a user of the CONVERGENCE System.
Versatile Digital Item (VDI)	A structured, hierarchically organized, digital object containing one or more resources and metadata, including a declaration of the parts that make up the VDI and the links between them.

3 Overview of CONVERGENCE system

The goal of the CONVERGENCE project is to enhance the Internet with a content-centric [3], publish-subscribe service model [19], based on a common container for any kind of digital data, including representations of people and Real World Objects. We call this container the Versatile Digital Item (VDI). VDIs are the basic unit of distribution and transaction in the CONVERGENCE network. The definition of VDIs is derived from the MPEG-21 Digital Item Declaration Standard [22].

VDIs can incorporate every possible kind of information, including signalling and control, and therefore minimize the need to store external information and states outside the data unit (though this is still allowed). The introduction of the VDIs corresponds to a shift from “host-centric” to “content-centric” networking, that is to a form of networking in which the network layer provides users with content, instead of providing communication channels between hosts, and is aware of this content, at least in the sense it knows its name. This shift is analogous to the switch from circuit to packet switching: in circuit switching a PCM slot contains only user data; in packet switching an IP datagram also contains (among other things) destination addresses. Similarly, in “content switching”, the VDI contains a complete package of user data and meta-data describing content and how to handle it.

The VDI container can be used to encapsulate any kind of digital information: not only classical media files, but also data about services, people and Real World Objects (RWOs) (e.g. items of merchandise identified with an RFID). VDIs bind meta-information (describing the content and structure of the item) and resources (other VDIs, audio, images, video, text, descriptors of RWOs, descriptors of people etc.). The meta-data describing the VDI includes structural information, describing the content of the VDI; cryptographic keys allowing robust authentication and protection of information included in the VDI; rights information defining rights to use the item; an expiry date, supporting “digital forgetting”. VDIs are identified by a unique identifier, which is translated (or which is equal) to a network-level name used to route the VDI.

The second key feature of CONVERGENCE is support for a publish/subscribe service model: subscribers register their interest in an event, or a pattern of events, and are asynchronously notified of events generated by publishers. Publish/subscribe effectively decouples the application end-points in space, time and synchronization. This allows for greater scalability, a more dynamic network topology and a much enlarged and flexible typology of services.

The main players in the CONVERGENCE framework are depicted in Figure 1: publishers advertise *resources* (data and service-access-points) on the CONVERGENCE system and *subscribers* express their interest in specific resources. The system notifies subscribers when the resources become available. Users can also *search* for resources and obtain an immediate response. In this respect *search* can be seen as a sub case of subscribe.

VDIs are used both to publish and to subscribe to content. Every resource that is stored or published in the CONVERGENCE system is associated with a VDI. Subscriptions express criteria that can be verified by inspecting VDI information. Therefore, CONVERGENCE system supports *content-based* subscriptions as defined in [19].

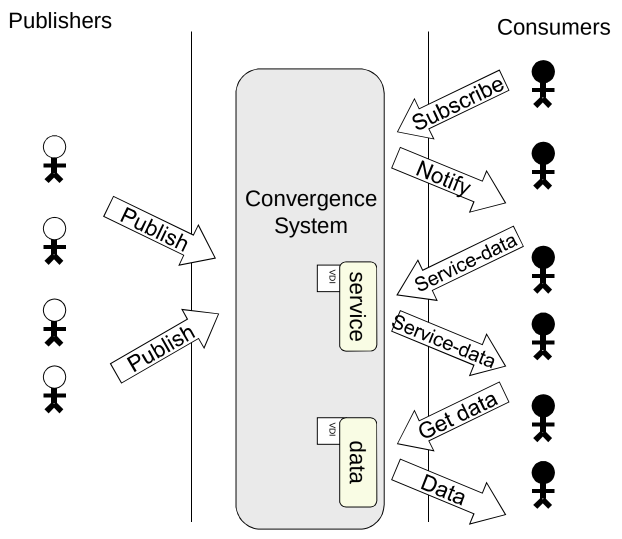


Figure 1: Service model

3.1 The CONVERGENCE Architecture

CONVERGENCE specifies an Information and Communication Technology (ICT) environment with the features described in this document. A CONVERGENCE system is an implementation of the specification consisting of a set of interconnected **peers** and **nodes** collectively called "CONVERGENCE devices".

Peers are based on a 3-level architecture. From the top down:

1. Application level (CoApp)
2. Middleware level (CoMid)
3. Computing Platform level (CoComp)

The Computing Platform level comprises key functional blocks providing novel content-centric networking (CoNet) and secure handling (CoSec) of resources within CONVERGENCE. The Computing Platform level also provides interfaces to access the local resources of the CONVERGENCE peers.

Nodes are devices that only include a CoNet networking component and/or a CoSec component, belonging as in peers, to the Computing Platform level. Therefore we can have CoNet nodes and CoSec nodes (see Figure 2).

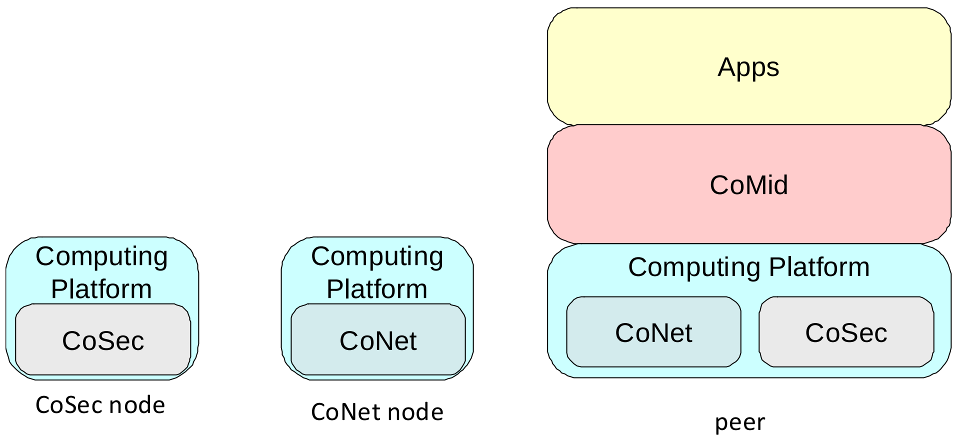


Figure 2: CONVERGENCE nodes and peers

Figure 3 depicts the three conceptual levels of the CONVERGENCE system architecture. Each level is delimited by the nature of what its components deal with. Figure 3 summarizes the scope of each level and the kinds of information exchanged at the interfaces between levels.

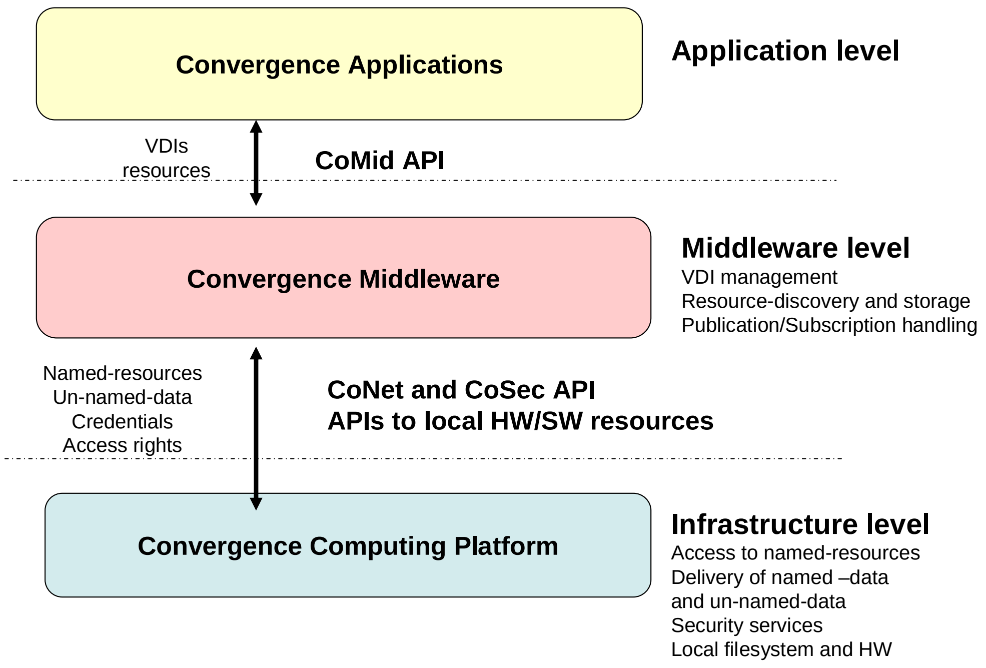


Figure 3: Overview of CONVERGENCE architectural levels

Figure 4 depicts the complete architecture of a peer. Each of the 3 levels has its own structure and communicates with other levels via standard APIs.

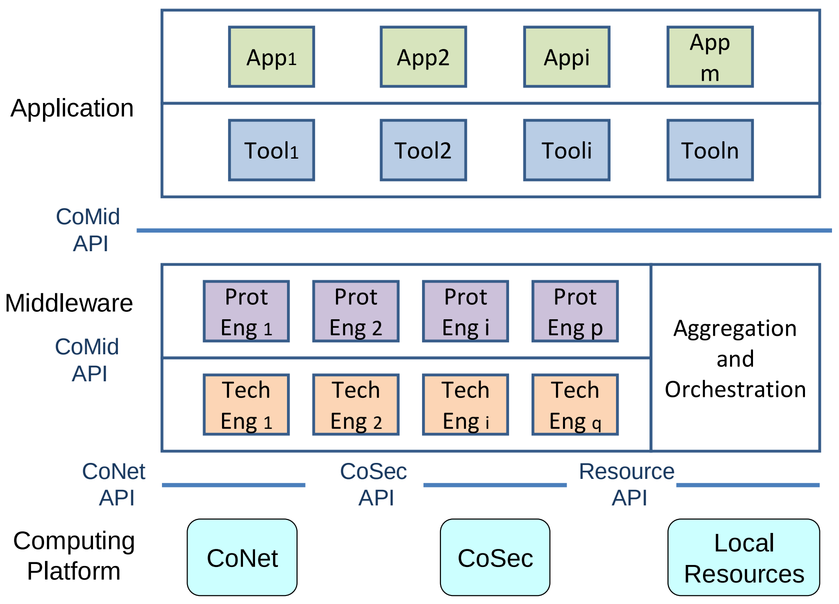


Figure 4: Architecture of a CONVERGENCE peer

3.1.1 Applications

Applications provide users of CONVERGENCE peers with the means to create, process and consume VDIs and their components, and thus to flexibly manipulate digital resources and deal with physical objects and their digital counterparts, as well as with people and their digital identities. These resources are produced and managed at the Application level. CONVERGENCE-compliant Applications use services provided by the CONVERGENCE middleware to create descriptors of resources in the form of VDIs. Hence resources and VDIs are exchanged at the interface between Applications and Middleware.

Users can download/install Applications to a peer. Applications can activate local and remote middleware functionalities, via the standard CoMid API. To expedite the development of Applications, CONVERGENCE defines special re-usable Application elements called Tools. Tools facilitate re-use of code in Applications, an Application can make use of several tools. The CONVERGENCE applications level can be seen as split into two sub-levels: a User Applications sub-level and a Tools sub-level. Therefore the term Application can be used in a wider sense to refer to both User Applications and Tools.

3.1.2 Middleware

The CONVERGENCE Middleware (CoMid) is the level responsible for creating, retrieving, manipulating and consuming VDIs. VDIs are published in the CoMid. The CoMid allows users to search for them using semantic search operations and delivers the results.

The CONVERGENCE Middleware level is based on the MPEG-M standard [21], which provides a distributed eco-system of Engines, i.e. components that can be activated by Applications running on MPEG-M compliant devices. CONVERGENCE builds on and expands the MPEG-M design. Specifically, CONVERGENCE introduces new Engines to:

- Implement the publish/subscribe paradigm on top of MPEG-M Event Notification services.
- Allow semantic searching of and subscriptions to published resources.

The CoMid is thus composed of Engines. Many are "native" MPEG-M engines, some are extended MPEG-M engines and a few are new CONVERGENCE-specific Engines.

There are two types of Engines:

1. Protocol Engines (PE) that activate functionalities in remote or local peers
2. Technology Engines (TE) that are typically called by PEs to execute specific functionalities.

Annex 1 provides a list of PEs and TEs included in CONVERGENCE along with the respective definitions.

An Application call may involve more than one PE/TE. Therefore CONVERGENCE provides a standard mechanism for "aggregating" PEs and "orchestrating" TEs. Work is under way to develop a single aggregation/orchestration technology applicable to PEs, TEs and, potentially to Applications as well. Engines communicate via the CoMid API.

Figure 5 depicts how an Application call may involve a chain of PEs/TEs. However, chains need not be linear: where useful, Applications can directly call TEs without PE intermediation.

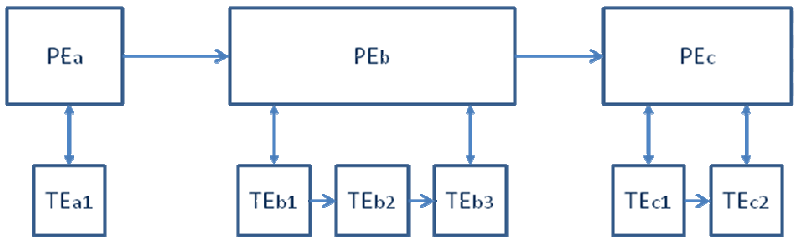


Figure 5: A chain of Protocol and Technology Engines

Figure 6 depicts how Peer 1 talks to Peer 2 via an internal PE, triggering the establishment of a chain of PEs/TEs.

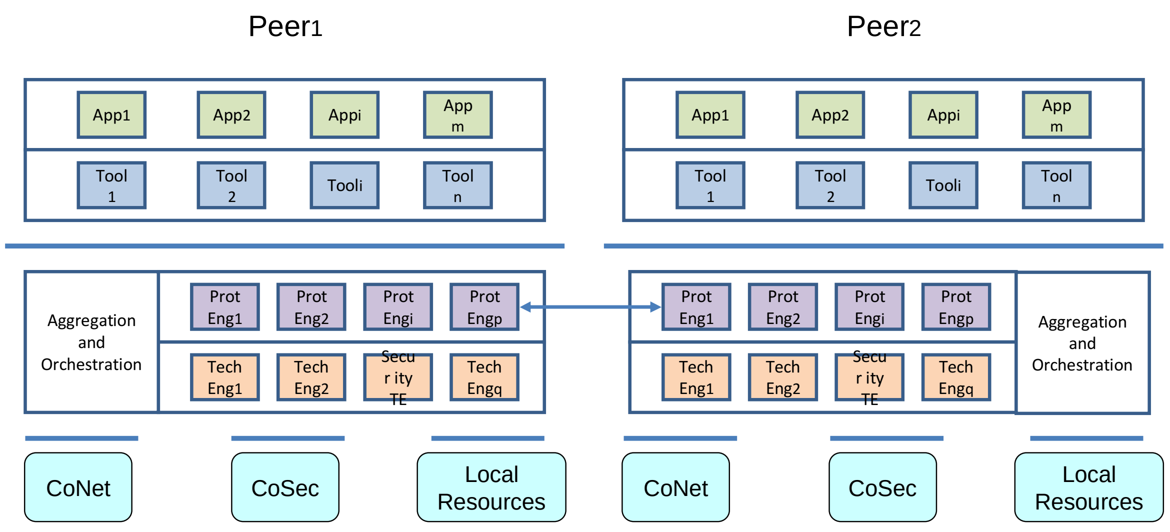


Figure 6: Communication between peers via PEs

CoMid provides the interface between Applications and the two key Computing Platform components: CoNet and CoSec, via their counterpart at the middleware level, the CoNet TE and the Security TE, respectively.

3.1.3 Computing Platform

The CONVERGENCE Computing Platform hosts specialized network and security modules, as well as interfaces to local resources, such as file-systems and processing power.

Engines access the local resources of the computing platform via a Computing Platform dependent API.

When resources are stored inside the network they became network named-resources (or interactive data sessions). The CoNet communication functionalities are accessed, at a low-level, via the CoNet API. Applications access these functionalities via the CoNet middleware engine.

Credentials and access-rights, extracted from Licenses embedded in VDIs, are exchanged with the CoSec components of the CONVERGENCE-enabled Computing Platform. Low-level security functionalities in the CoSec block are accessed via the CoSec API. Applications access these functionalities via the Security middleware engine.

3.1.4 A distributed view of the CONVERGENCE system

Figure 7 provides a distributed view of the CONVERGENCE system, seen as a set of interconnected peer and CoNet/CoSec nodes. Application functionalities (and related security) are provided in a distributed way by CONVERGENCE peers. Network level functionality and security functionality are realized in a distributed way exploiting support of the set of CoNet/CoSec nodes.

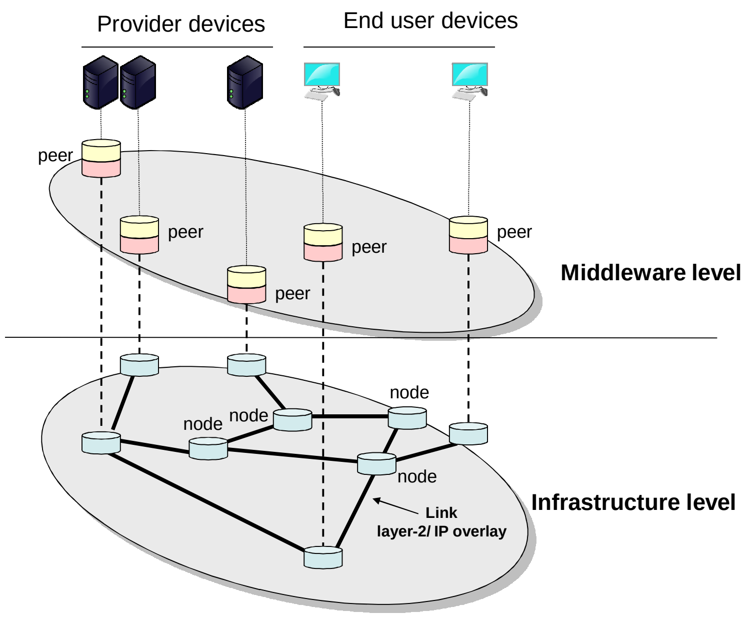


Figure 7: Distributed view of CONVERGENCE system

3.2 CoMid features

The following sections briefly introduce key CoMid functionalities, making it possible to:

- Operate on VDIs, describing resources with metadata extracted from well-known or custom semantic taxonomies of concepts;
- Publish information on resources into an overlay of peers whose topology is based on those same semantic taxonomies; this overlay will be referred to as Semantic Overlay
- Search for resources in specified regions of the Semantic Overlay and deliver matching content to users, even when resource descriptions do not fully match user requests.

3.2.1 The VDI

CONVERGENCE users act on Versatile Digital Items (VDIs). These are XML structures containing:

1. Identifiers
2. (links to) Resources
3. (links to) Semantically-rich metadata describing resources
4. (links to) Licenses expressing which rights are given to act on resources
5. Event report requests (ERR) instructing a peer to issue an event report (ER) to a specific user/peer in the event certain actions (e.g. play, store or match) are performed on a resource

So far, CONVERGENCE has defined 4 types of VDI: Resource (R-VDI), Publication (P-VDI), Subscription (S-VDI) and User (U-VDI).

As part of the creation process, a VDI is assigned a unique and persistent identifier. When a VDI is "superseded" by a new version or when a new VDI is related to an existing one, the CONVERGENCE Ontology Services let users establish links between them.

3.2.2 Semantic and Dictionaries

The CONVERGENCE Core Ontology (CCO) is a native CONVERGENCE component. The CCO allows for a semantic organization of peers in a virtual overlay network of "fractals", which are dynamically shaped and connected, on the basis of users' interests in different types of content. Peers join or leave a fractal based on what users are currently publishing and subscribing to. The basic structure of the CONVERGENCE Semantic Overlay is shown in Figure 8. A peer typically belongs to more than one fractal, depending on how many users are interested in what kind of content. In order to provide redundancy and mitigate peer churning (peers may be offline when users wish to access them) fractals are usually populated by more than one peer.

Domain and user ontologies can also be used to define fractals. Typically the former come from Service Providers (SP), while the latter are created by the individual users. Therefore, the fractal organization can be seen as covering multiple "dimensions", where each dimension is defined by an ontology. Thus, a peer may reference resources described using concepts based on different ontologies in different dimensions.

The Community Dictionary Service (CDS) component is also part of the middleware and is implemented as a Technology Engine (CDS TE). This component maintains *dictionaries* that help translate concepts and properties from one ontology model to another. The CDS is exploited when users describe resources and when descriptions of what is being published do not match the terms used in user definitions of search criteria. In this cases, the CDS attempts to translate betweens concepts and resolve the match.

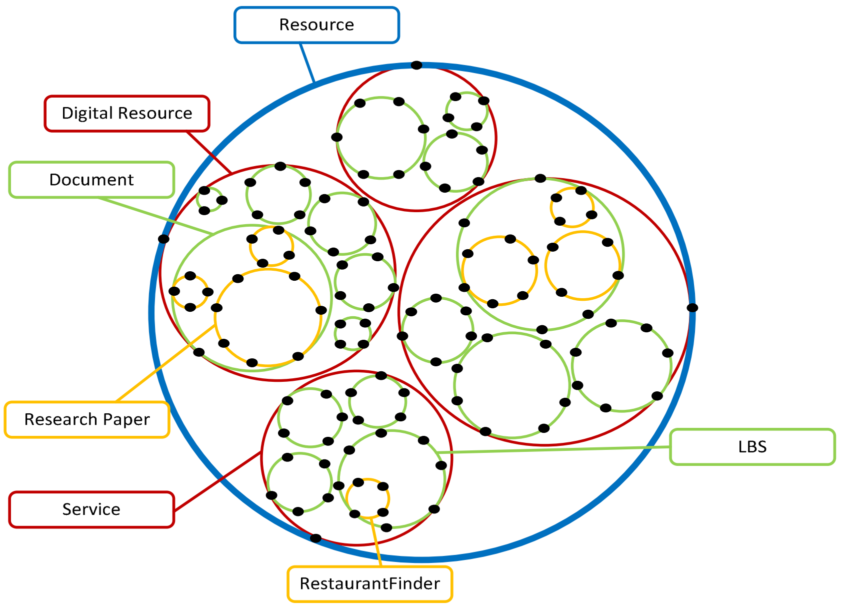


Figure 8: An example of structure of CONVERGENCE fractals

3.2.3 Publish/Subscribe

The key interaction pattern for CONVERGENCE users is based on a publish/subscribe paradigm. Users who make resources available to the system and discoverable are said to publish them. Users searching for specific resources are said to subscribe to them. The interaction is asynchronous and decoupled in time and space. Publications and subscriptions are both described by semantically-rich metadata and labelled with relevant concepts. To find relevant content, a dedicated CoMid engine executes a semantic query, representing the subscription. The query is successful when published content matches the subscription criteria.

Fractals in CONVERGENCE Semantic Overlay represent focused concepts, i.e. unions and/or intersection of concepts in the core ontology or other ontologies (dimensions). All P-VDIs and S-VDIs (which carry publications and subscriptions) labelled with a certain concept, are stored in a specific set of peers in the fractal. Spatial redundancy (e.g. using more than one peer to store information) is used to ensure propagation of information, because at any given time some peers might be inactive. The use of “focused” fractals makes it possible to restrict the search space to peers actively participating in the fractal.

Peers have the embedded ability to:

1. Perform matches between P-VDIs and S-VDIs
2. Communicate any match to specified peers in the form of ER (Event Reports), depending on licenses and ERRs (Event Report Requests)

3. Remove S-VDIs and P-VDIs from the match tables when
    a. Their expiration date has passed
    b. An authorized user requests to remove them before the expiration date
4. Aggregate ERs from different peers and communicate the result to the end user

3.3 CoNet features

CoNet features are **very** briefly recalled in this section; we urge interested readers to read the section 7.1, which gives a comprehensive vision of this important aspect of the CONVERGENCE system (or at least the first 5 sub-sections: 7.1.1 through 7.1.5).

The retrieval of resources and the communications between peers is made possible by a content-centric network, called CoNet that lets users access remote named-resources (as opposed to remote hosts as in current Internet).

As Figure 9 shows, *named-resources* can be:

1. *Named-data*: i.e. a sequence of bits, like a VDI or the resource the VDI refers to
2. A named-service-access-point (*named-sap*): i.e. a network endpoint from which a Protocol Engine (or any other kind of service entity) receives CoMid messages.

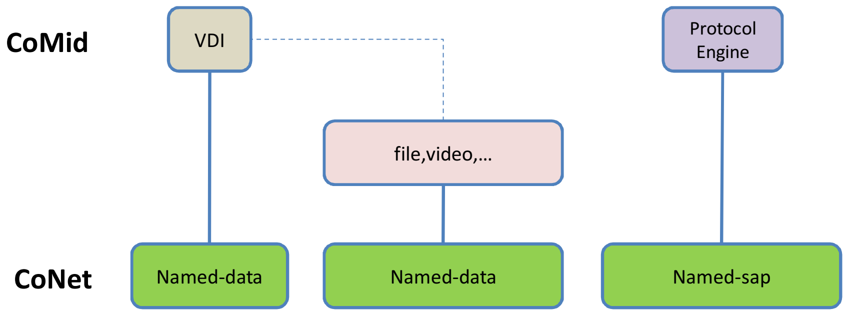


Figure 9: Mapping among CoNet named-resources and middleware entities

In both cases, the named-resource is identified by a *network-identifier*, i.e. a *name* like "foo:VDI1". In the current CONVERGENCE implementation, network identifiers coincide with the identifiers assigned to VDIs and their components by a CoMid service.

Note that the CoMid level and CoNet component in the Computing Platform level are completely decoupled: a VDI is a CoMid data-unit. For the CoNet a VDI is simply a sequence of bits addressed by its network-identifier. The CoNet has its own data-unit, defined later on (see 7.1).

CoNet is aware of the network location of named-resources and uses *routing-by-name* to route the request to the copy of the named-resource held by the network node that is closest to the requesting user. The copy could be in a cache or in a serving-node (i.e. a server). Compared to traditional content-centric architectures (e.g. CCNx), CoNet limits the size of name-based routing tables by including only a subset of all named-resources. Missing entries are looked up in a name-system and then cached. Moreover, CoNet does not keep states in network nodes to deliver the actual resource from a serving-node (or a cache) to the requesting peer. We believe that these features will improve network scalability with respect to the number of named-resources and the number of ongoing communications.

CoNet supports *built-in caching/replication* functions. To improve access to popular resources, the same named-resource can be replicated in different network nodes. CoNet provides users with access to the most convenient replica. Replica nodes could be pre-provisioned, as in Content Delivery Networks, or opportunistically selected by *in-network caching* mechanisms. In-network caching prevents denial of service boys called "flash crowds", i.e. situations when a very large number of users simultaneous access a popular resource. Unlike state-full and off-the-shelf transparent proxy technologies, CoNet performs stateless caching. This speeds up caching and reduces the cost of implementation.

CoNet provides *content-based quality of service*. Network nodes can differentiate performance in terms of bandwidth and storage (caching) on the basis of the name of the resource they are serving. For instance, the named-data "foo:VDI-high-priority" might have a higher transmission priority and a higher probability of being cached locally than the named-data "foo:VDI-low-priority". Unlike current IP technology, content-based QoS mechanisms do not require complex and slow deep packet inspection (DPI).

CoNet also handles *digital forgetting.* The owner of named-resources may request CoNet to remove a named-resource from all serving-nodes and caches. This can be achieved either by specifying an expiration time for the request or by making an explicit request for its removal.

3.4 CONVERGENCE Security and CoSec

3.4.1 Overview

Security is an essential feature of CONVERGENCE. The system's main security features are: i) assurance of VDI integrity (and authenticity); ii) governance of VDI access restrictions (confidentiality); iii) user identification and authentication; iv) issuing and enforcement of licenses; v) protection of user privacy; and vi) network security.

Most of these features are provided at the middleware level (CoMid), some at the Computing Platform levels. CoMid security features are provided by the Security TE, which in turn relies on features offered by CoSec in the Computing Platform. Some security features of the CoNet are provided by CoNet itself; others require support from the CoSec.

From a methodological standpoint, the CONVERGENCE project defines *security assets*, considers *threats* to these assets, and derives *security functional requirements.* These requirements then drive the selection and development of appropriate cryptographic primitives and protocols. This methodology, which we apply to each of the issues described above, is described in greater detail in sections 4.14 and 4.15.

A key task of the methodology is to identify the "roles" that lie at the base of CONVERGENCE security. Note that this task is logically prior to the identification of the architectural entities (Technology Engines and Protocol Engines) that will implement the required functionality in the CONVERGENCE architecture (see section 3.1.2).

Two key roles are those of the *Identity Provider* (a trusted third party responsible for the *registration* of users, identification of users, verification of credentials, and issue of certificates) and the *Service Provider*, the entity that handles "daily" business in a specific scenario, including user authentication and licenses. One of the benefits of separating the two roles is *privacy protection*: identity Providers never gain access to data accumulated by Service Providers. Service Providers do not need to know all (or any) of the personal data Identity Providers collect during registration. A typical *Identity Provider* might be a *government agency* that performs end-user registration and supplies Service Providers (e.g. *insurance companies*) with users' certified credentials.

3.4.2 CoMid Security

The CoMid exploits the features offered by the CoSec component in the Computing Platform through a dedicated Technology Engine called the Security TE. Components wishing to perform a security function or protocol will use CoSec via the Security TE. The CoSec serves all requests coming from the other engines and directed to the Security TE.

Based on CoSec, the Security TE can: i) create new credentials and manage certificates; ii) generate keys and encrypt/decrypt data or keys; iii) store confidential information (e.g. licenses and keys) in the secure repository; iv) certify the integrity of engines.

Other Engines rely on the Security Engine to perform the following operations (see Figure 10): i) signing of VDIs (VDI TE); ii) symmetric encryption/decryption of resources (Media Framework TE); iii) asymmetric encryption/decryption of a key (REL TE); iv) user Identification (Identify User TE); v) user authentication (Authenticate User TE).

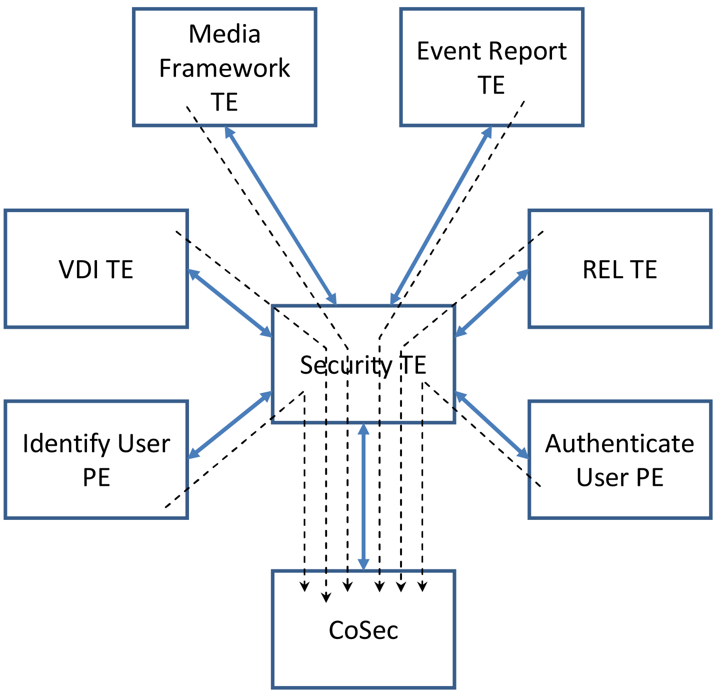


Figure 10: Security for CONVERGENCE CoMid

Secure repositories play a particularly important role in the CoSec security architecture. The preferred option for implementing secure repositories in CONVERGENCE will be a smart card. Unlike a piece of software, a smart card is highly tamper-resistant. Smart cards also have the ability to *process* security relevant protocols (e.g. issue and validate signatures, validate certificates, generate key pairs, etc.), providing functionalities far beyond secure storage.

3.4.3 The CoSec

The *CoSec* component in the Computing platform level is responsible for handling the majority of cryptographic protocols and security related tasks. Although the CONVERGENCE architecture diagram shows it as a single monolithic block, it actually has a *distributed architecture* encompassing several independent (and possibly distant) components, each of which includes software as well as hardware.

Most components of CoSec are located on *client computers* (e.g. end-user laptops), *smart cards*, *application servers* and *network peers*. The majority of protocols processed within CoSec involve *several* of these entities. The figure below illustrates the general style of these protocols as exemplified by a simplified user authentication protocol. In the example, the *smart card* acts as a *secure repository* for the user's signature private key.

CoSec features are exposed to the CoMid components (engines) through the CoMid Security TE. A CoSec API is defined so that the Security TE can use the CoSec. The CoNet can access CoSec features through the CoSec API.

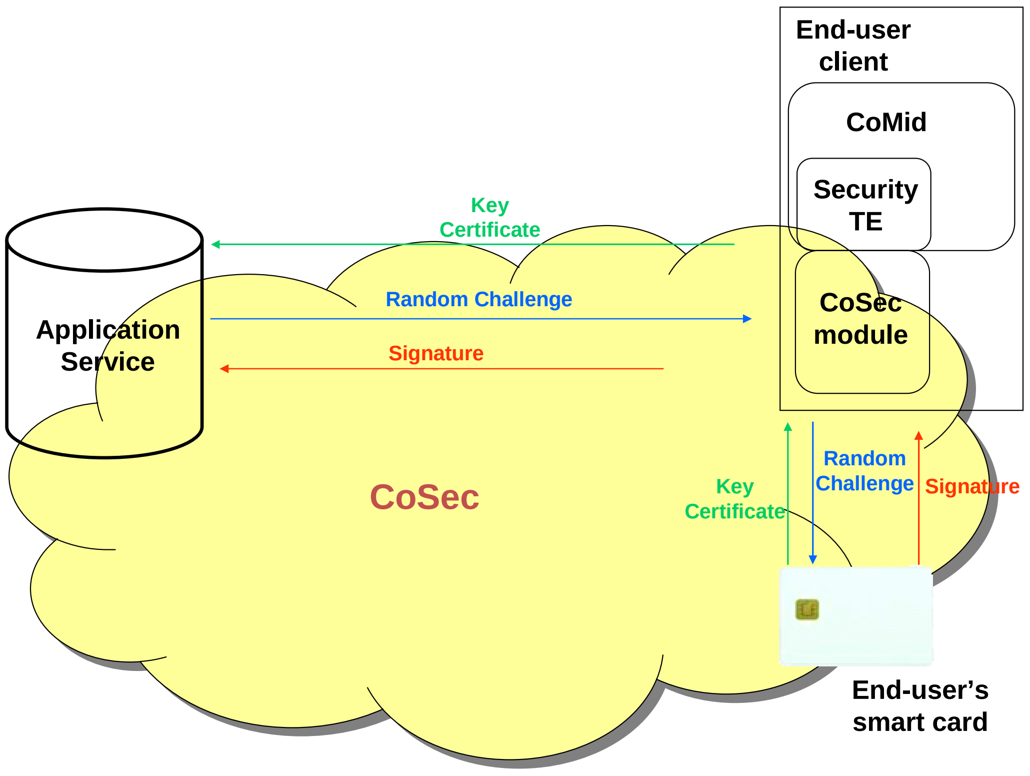


Figure 11 CoSec elements and example information exchange

3.4.4 CoNet security

CoNet supports security and privacy mechanisms aimed at preserving the integrity of the networking service and, where required, the anonymity of owners and consumers of named-resources. A distinguishing aspect of CoNet security is the use of *data-centric security*: security information is embedded in CoNet data-units. Data-centric security makes it possible for user and network nodes to verify the validity of named-resources, avoiding the caching and dissemination of fake versions. Protecting information at the source (i.e. protecting the data unit) is more flexible and robust than delegating this function to applications, or securing only the communications channels. CoNet security operations may be supported by CoSec.

3.4.5 Cryptographic primitives and protocols in CoSec

CONVERGENCE will use various cryptographic protocols to implement security requirements, including both off-the-shelf solutions and new primitives which are subjects of ongoing research.

Fast *symmetric encryption* and *decryption* of content (off the shelf): this will use algorithms like AES-CBC; key escrow may be standard or based on sophisticated protocols such as ABE (Attribute Based Encryption), IBE (Identity Based Encryption)
Asymmetric cryptography (off the shelf): this will use established primitives like RSA or Elliptic Curves; asymmetric cryptography will be used for key agreements, signatures, certificates, key wrapping, etc.
Basic primitives like cryptographic hashes - (off the shelf)
Group Signature Protocol – (current research): group signature protocols will allow a member of a pre-specified group to anonymously sign a VDI's content or a challenge during authentication. It will only be possible to break anonymity on request (e.g. by a government entity as it may be requested by Law)
Identity and Attribute Based Encryption - (current research): in this scheme recipients of content (or messages) are assigned specific (arbitrary) attributes; the provider of content can encrypt according to these attributes, so that the possession of the same attributes is needed to decrypt the content.
Pseudonymous access via "Restricted Identification" - (current research) Within a specific context, each user identifies herself with a unique pseudonym; for disjoint context, users' pseudonyms cannot be linked.

Section 7.2.2 will provide more details on the adopted cryptographic primitives and protocols.

4 Main technical and architectural concepts

4.1 The MPEG-M standard

The emerging MPEG-M standard (ISO/IEC 23006 – Multimedia Service Platform Technologies) is a suite of standards providing Architecture, Technology Engines (TE), Protocol Engines (PE), Aggregation Technologies and Reference Software that are well aligned to the basic concepts of CONVERGENCE as represented in Figure 3 (Overview of CONVERGENCE architectural levels). The standard includes:

- Part 1 Architecture
- Part 2 MPEG Extensible Middleware
- Part 3 Reference Software
- Part 4 Elementary Services
- Part 5 Service Aggregation

The figures representing the peer architecture depicted above in this document are based on MPEG-M. Part 2 introduced the notion of Engine, i.e. an appropriate groupings of technologies and the Application Programming Interfaces (API) through which an Application can access the functionalities it needs. As an Application typically needs more than one engine ("chains" of engines), MXM also provides examples of Orchestrator Engines, special MXM Engines capable of creating chains of Engines to execute, high-level application calls such as "Play" (see Figure 12).

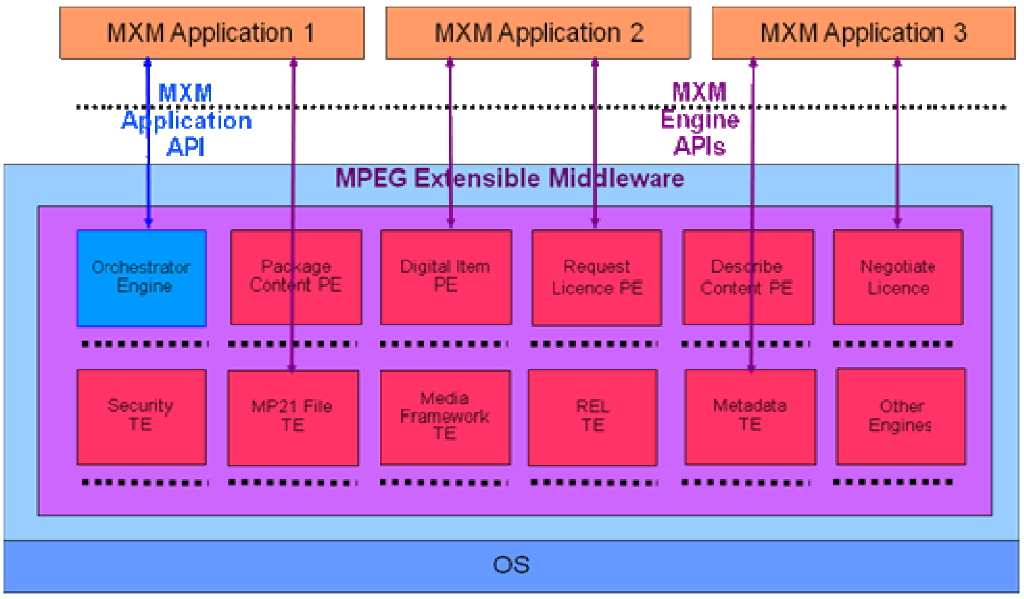


Figure 12 – The MXM standard

Part 4 introduced the notion of Elementary Services and specifies the messages exchanged between two peers. An implementation of protocols as specified by Part 4 is called a Protocol Engine. Finally Part 5 specifies how Protocol Engines can be chained to provide so-called Aggregated Services.

In recent years, many new digital media related services have appeared. Many such services are actually combinations of Elementary Services. MPEG has seen that, standardizing a set of technology elements and communication protocols facilitates the creation of aggregated services, from a set of standard Elementary Services, even on demand.
Assuming that in there is a Service Provider (SP) for each Elementary Service, a User may ask the Post Content SP to get a sequence of songs satisfying certain Content and User Descriptions. The Figure 13 below depicts how a chain of Services can respond to the User's request.

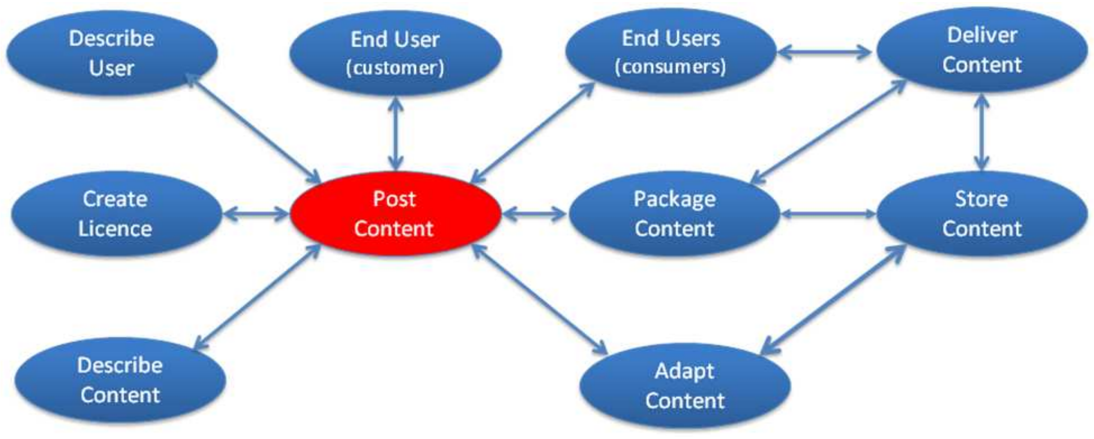


Figure 13– A possible services chain centred around Post Content SP

End User contacts Post Content SP who gets appropriate information from Describe Content SP and Describe User SP to prepare the sequence of songs using its internal logic. She then gets the necessary licenses from Create License SP. The sequence ("titles") of songs is handed over to Package Content SP. Package Content SP gets the songs ("Resources") from Store Content SP and hands over the Packaged Content to Deliver Content SP which streams the Packaged Content to End User.
MPEG has specified a set of standard Elementary Services and related protocols to enable distributed applications to exchange information about entities playing a role in digital media services (e.g. Content, Contract, Device, Event and User), and the processing that a party may wish to execute on those entities, (e.g. Authenticate, Create, Deliver, Describe, Identify, Negotiate, Process, Request, Search and Transact). These have been standardized in part 4 "Elementary Services" [31].

Given these advantages, CONVERGENCE has decided to adopt MPEG-M as the basis of its middleware architecture, including MPEG-M protocol and technology engines, and aggregation technologies. Many CONVERGENCE Protocol and Technology Engines come from MPEG-M.

CONVERGENCE extends the Describe Content PE to support integration with ontologies (CDS).
Specific CONVERGENCE functionalities are supported by other new engines:

Protocol Engines

1. Describe Content (extended functionality)
2. Inject Content (new)

Technology Engines

1. Community Dictionary Service (CDS)
2. CoNet
3. Match
4. Overlay
5. Security (extended)

CONVERGENCE is developing and testing its new engines, and plans to propose a selection of them for standardization.

4.2 Content Centric Networking

Several papers (e.g. [11][15][7][16]) and research projects ([8][9]) propose a shift from “host-centric networking” to “information centric” or “content-centric” networking. The essence of Content-Centric Networking (CCN) is that the network layer provides users with content, instead of communication channels between hosts, and is aware of the content, at least in the sense of knowing its “name”. A CCN architecture should:

- Address contents, using an addressing scheme based on names, which do not include references to their location;

- Route a user request, which includes a content-name, toward the closest copy of the content with such a name (name-based, anycast routing) and deliver the content to the requesting host;

- Provide native, in-network caching functionality to achieve efficient content delivery both in fixed and mobile environments [17];

- Exploit security information embedded in the content to avoid the diffusion of fake versions of content and to protect the content (a more robust solution than entrusting security to applications, or securing only the communications channels [16]);

- Provide a way to differentiate the perceived quality provided by different services [18], and provide per-content quality of service differentiation, covering cache hits.

Network level functionalities in CONVERGENCE are provided by the CONVERGENCE Network (CoNet) -a content-centric inter-network that provides users with network access to remote named-resources over a public or private network infrastructure [1][2][3].

In the CONVERGENCE CoNet, named-resources can be either data[1] ("named-data") or service-access-points[2] ("named-service-access-points"), identified by a network-identifier (a name).

As shown in Figure 14, CoNet interconnects CoNet Sub Systems, which can be layer-2 networks, layer-3 networks or couples of nodes connected by a point-to-point link. CoNet supports the "clean-slate" and "overlay" approaches to deployment, already proposed in the literature. In addition, CoNet supports a novel "integration" approach, which extends the IP layer with a new header option that makes IP content-aware[6].

CoNet limits the size of name-based routing tables by including only a subset of all named-resources; missing entries are looked up in a name-system and then cached. CoNet does not maintain states in network nodes during delivery of content.

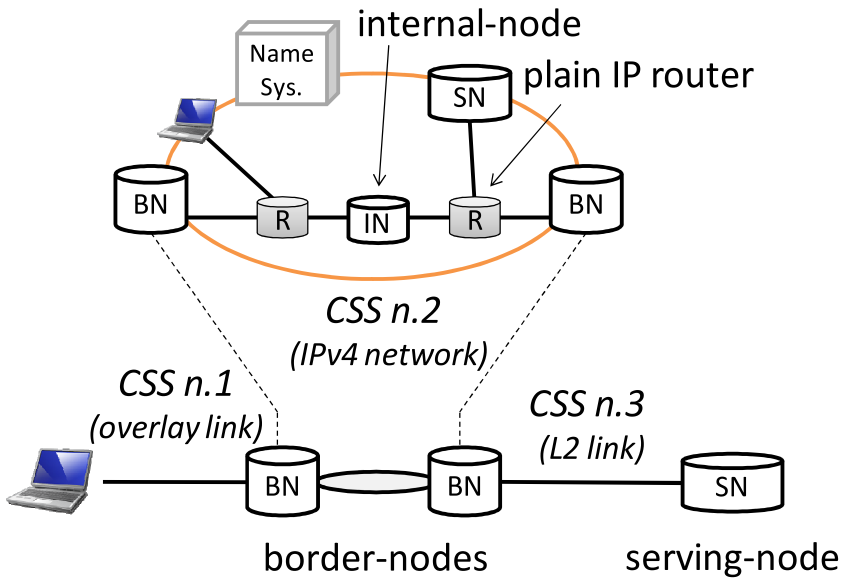


Figure 14: CoNet Architecture

[1]Named-Data includes: documents, video, images, structured information, VDIs. The Network level is general purpose, in the sense that it can handle any kind of data and not only data generated by the CONVERGENCE system.

[2] A named-service-access-point is a network endpoint through which an upper layer entity (e.g., a server or a client) sends and receives data. In the current Internet, for instance, TCP port n. 80 is the default service-access-point for HTTP servers.

4.3 VDI

The basic design of CONVERGENCE makes a functional distinction between different kinds of VDIs. CONVERGENCE's basic building block is a **Resource –** the bits representing a multimedia stream or the digital representation of a real-world object, for instance.

As a first conceptual step we create a **Resource VDI** around the resource, and let users store it somewhere in the network.

The second conceptual step is to publish the resource to the world. To achieve this, the system uses a generic gossiping protocol over an overlay of peers (the CONVERGENCE Semantic Overlay). The information to be gossiped is gathered from the Resource VDI, packaged into a transport package and gossiped from peer to peer. The transport package is also a VDI. We call it a **Publication VDI**.

The reason we use VDIs to advertise the publication of another VDI is because it is convenient: all the information that needs to be gossiped (metadata, licence, event reports) fits naturally in a VDI and the gossiping protocol does not care about what it is transporting. However, the most important reason is that when information about a publication is packaged as a VDI, it can be stored in the physical network in the same format, independent of the overlay. By observing the flow of VDIs across the network, external entities that know the VDI schema can collect statistics about what is being published and subscribed to on the system.

Publication VDIs offer an additional level of indirection between the Resource and the Resource being advertised, an important advantage in complex business scenarios. Publication VDIs allow users to publish the same resource, with different metadata, at different times, using different identities. For instance, a "second publisher" with appropriate permission from the author, can also publish the resource – perhaps because he wants to publish it in a different context. All he needs to do is provide a new Publication VDI, pointing to the same Resource VDI. The CONVERGENCE system tries not to restrict users' ability to invent new patterns of business so long as they hold the necessary permissions. Publication VDIs carry a licence and an expiry date.

Subscription VDIs, issued by users searching for specific resources, play a role symmetrical to that of Publication VDIs. They carry a licence and an expiry date, too.

4.3.1.1 Independence of Publication VDIs from Resource VDIs

The introduction of Publication and Subscription (Pub/Sub) VDIs decouples the functionalities of the middleware and the network level. CoMid is all about matching publications and subscriptions, and discovery, and basically does not deal with content. To use an analogy with today's Internet, it is the Google database. CoNet, by contrast, is equivalent to ISP servers storing web pages. The two levels are independently managed. This decoupling makes the system more robust.

Since the CoMid and CoNet are decoupled, the only point in common between a Resource VDI and its associated Publication VDI is the VDI identifier that links them. This is because they serve two completely different purposes. A Resource VDI represents a "thing"; a Publication VDI represents information about the "thing".

In principle, a Resource VDI and its associated Publication VDI can have distinct expiry dates. The expiry date of the Publication VDI determines how long the resource will be discoverable. When it expires, the Resource VDI will no longer show up in searches and subscriptions (it will no longer be visible at the CoMid level). However, it will still be possible to fetch it using a direct link held in the CoNet level. In many practical situations, the Resource VDI will not have an expiry date. In other cases, the Resource VDI may expire before the Publication VDI referring to it. This makes it possible to conserve information about a resource, even when the resource itself is no longer available. Resource VDIs can circulate in the system without being associated with a Publication VDI. This means they can be indexed by crawlers but are not part of the pub/sub process. Expiry of such resources is managed by CoNet expiry mechanisms, which are independent of mechanisms in CoMid and work independently of whether they have been published.

CONVERGENCE is designed to facilitate automated generation of Publication VDIs from Resource VDIs. If publisher and author are the same person (if the principal in the two Licenses is the same) the process can be fully automated. In the default scenario, the two VDIs are given the same expiry date, though users can configure different expiry dates if they wish. If publisher and author are different persons (the system can tell), the Publish Aggregated Service will ask the publisher whether she wish to keep the licensing conditions defined in the Resource VDI or define new ones.

4.3.1.2 Storage of Publication and Subscription VDIs

Storing Publication and Subscription VDIs on the network has important advantages: by using a content-centric paradigm and a standard container for publications and subscriptions, the format of the database that indexes resources is well-known, and the database itself is highly available (though visibility of some VDIs may be restricted by License).
Possible strategies for assigning Publication and Subscription VDIs to networks nodes include storing them in the same peer where they have been created, or on the first peer they are gossiped to, (providing the peers deploy the full CONVERGNCE stack, down to the CoNet level) The project has not yet made a final choice. However, it is a clear requirement that they should be reachable from CoNet, using well-specified names.

4.3.1.3 User VDIs

The project is exploring the possible introduction of another kind of VDI, which we call the **User VDI**. This allows for a level of indirection between CoMid services and the real location and/or identity of the end-user accessing services. This concept will be further explored and refined in future deliverables.

4.3.1.4 The Unpublish and Unsubscribe VDIs

The project is currently studying the definition of UnPublish and UnSubscribe VDIs. Given that peers will join and leave the system (as they go on or off line), in an arbitrary fashion, it is not possible to disseminate information about the revocation of a Publication or Subscription VDI to all peers in a synchronous and fail proof way. Whatever information spreads across the system at a specific instant, there will always be peers that miss it, because they are off-line.

To deal with this issue, Unpublish and Unsubscribe commands must be distributed asynchronously. The injection of UnPublish and UnSubscribe VDIs into the system signals a specific command (unpublish or unsubscribe) which is asynchronously gossiped as peers come online and go offline. This way, the information is disseminated to all peers as soon as possible.

4.4 *VDI dynamics and linking*

Current systems for versioning of on-line digital content and for creating links between content rely on rigid, semantically mute or ambiguous connections between digital objects. More specifically, the only standard and universal (HTML based), way of declaring a connection between two digital documents is to insert the URL for one of them in the other document. The usual way of doing this is to use *<a>* or *<link>* tags, indicating that the referenced document is somehow related to the referencing one. The specification of the tags provides some support for the expression of the nature/type of these relationships (3.4.2.1 [20]). Nonetheless, the number of types that can be expressed is very low and the types themselves are semantically ambiguous. Hardly any of the major on-line content retrieval and access systems (browsers) use the information.

CONVERGENCE aims to change this situation by providing users with a universal, explicit, rich and semantically unambiguous way of declaring and exploiting relationships between published digital objects (VDIs). In this way, CONVERGENCE will provide interesting new features.

- Clear, universal and unambiguous declarations of versioning relationships between VDIs will make it possible to group them into sequences of subsequent VDI versions and provide a universal and explicit mechanism for updating digital documents.
- Universal and unambiguous declarations of a base set of logical relationships between VDIs will enable users to declare a VDI to be a comment or correction to another VDI, and allow the system to automatically and unambiguously detect and process such relationships.
- Context dependent declarations of logical relationships between VDIs (or between sections of VDIs) will allow users to define “application scope” or “VDI scope” relationships, which can be interpreted using a domain-specific ontology.

4.4.1 Functional Requirements

CONVERGENCE supports two main mechanisms for expressing the relationship between VDIs.

- The first mechanism consists of a declaration that a VDI belongs to sequence of VDIs in which each VDI in the sequence is derived from the previous node in the sequence. In this way the first node in the sequence is the initial VDI and the last node is the most recent one.

- In the second mechanism, every VDI contains a declaration, inside the VDI, of its logical relationships to other VDIs. For example, if VDI B is a correction to VDI A, VDI B will contain an explicit declaration that VDI B maintains a relationship of "*correction*" to VDI A.

The implementation of these mechanisms is described in D4.1, sections 5.2, 5.3 and 6.3.

4.5 CDS

The Community Dictionary Service (CDS) is the CoMid component responsible for supporting CONVERGENCE semantic functionalities. To support this role, the CDS maintains and exploits ontologies and the relationships between them. CDS is a distributed service: each peer has access to a local CDS which might be customized with user ontologies, and/or can use more complete CDSs to be accessed remotely.

The following sections briefly describe how the CDS supports semantic descriptions of VDIs, and the way these descriptions are exploited to provide semantic matching between publications and subscriptions.

4.5.1 CDS supports content description

Given Convergence's content-centric publish/subscribe paradigm (see section 4.10), it is vital to correctly match published VDIs to user subscriptions. This requires coherent semantic descriptions of VDI resources and of users subscription criteria. Users employ CDS servers, which contain ontology models that can accurately describe the resource they wish to publish and to subscribe to. Each user can run her own (local or remote) CDS server, loaded with custom/users ontologies as well as well-known domain ontologies. The following paragraphs describe how the CDS supports this process.

4.5.1.1 CDS involvement in publishing

To help users in building semantic descriptions of their VDIs, the CDS exposes a service for ontology entity exploration. The service allows users to select ontology entities to semantically describe their VDIs. The walkthrough below describes the creation of the semantic description of a VDI for the movie "Star Wars". The user's CDS is loaded with the Movie Ontology (www.movieontology.org) and the IMDB ontology [40]. The identifier of the VDI is set to RVDI_23.

Step	User action	System response	Semantic description
1	User types "movie"	CDS searches for an entity named movie and returns the imdb:Movie	

		and the movieOntology:Movie Classes	
2	User selects movieOntology:Movie	System selects the movieOntology ontology for the description of the VDI and returns the rdf description	<rdf:rdf> <rdf:description rdf:about=”RVDI_23”> <rdf:type rdf:resource=”&movieOntology;Movie”/> </rdf:description> </rdf:rdf>
3	User types title	CDS returns with the movieOntology:title DatatypeProperty	
4	User selects movieOntology:title	System prompts user to enter the title of the movie	
5	User types “Star Wars”	System returns the user the rdf description	<rdf:rdf> <rdf:description rdf:about=”RVDI_23”> <rdf:type rdf:resource=”&movieOntology;Movie”/> <movieOntology:title>Star Wars</movieOntology:title> </rdf:description> </rdf:rdf>

4.5.1.2 CDS involvement in subscribing

The same service can help users to define subscription criteria. The walkthrough below presents the creation of a SPARQL query for the movie “Star Wars”. User’s CDS is again loaded with the Movie Ontology and the IMDB ontology.

Step	User action	System response	Semantic query
1	User types movie	CDS searches for an entity named movie and returns the imdb:Movie and the movieOntology:Movie Classes	

2	User selects imdb:Movie	System selects the imdb ontology for the description of the VDI and returns the sparql query	SELECT ?x
			WHERE
			{
			?x rdf:type imdb:Movie
			}

3	User types title	CDS returns with the imdb:title DatatypeProperty	
4	User selects imdb:title	System prompts user to enter the title of the movie	

5	User types "Star Wars"	System returns the user the SPARQL query	SELECT ?x
			WHERE
			{
			?x rdf:type imdb:Movie
			?x imdb:title "Star Wars"
			}

4.5.1.3 Matching

Matching is one of the core issues that the CONVERGENCE system has to face. The plethora of ontology models and the freedom granted to users to select the ontology with which they describe their VDIs (or to use their own ontologies) inevitably creates a huge diversity in the metadata used to described VDIs, and the risk that user publications and subscriptions will not match unless they are both described using the same ontology.

4.5.2 CDS dictionaries

4.5.2.1 Introduction

CDS dictionaries are a new concept designed to address the issues of heterogeneous metadata. In essence, a CDS dictionary is an ontology, just like user ontologies and domain ontologies. Unlike other ontologies, however, its whole content consists of a mapping between two different ontologies: that is, a set of equivalence statements between entities in the first ontology and entities in the second. In this way, CDS dictionaries provide semantic bridges between user-ontologies, between user-ontologies and domain ontologies or between domain ontologies.

The figure below depicts the set of ontologies the CDS uses during publication, subscription and matching procedures.

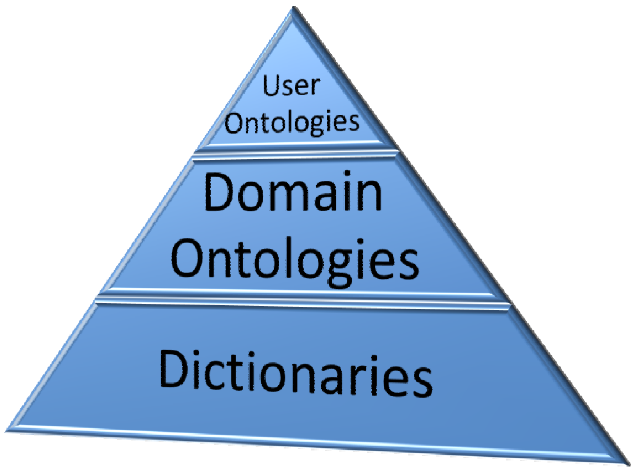


Figure 15 - CDS Ontologies

4.5.2.2 CDS dictionary format

As an example of the CDS dictionary format, the table below provides a mapping between the movieOntology and the imdb ontologies.

movieOntology ontology	`<owl:Class rdf:about="&movieOntology;Movie"/>`
	
	`<owl:DatatypeProperty rdf:about="&movieontology;title">`
	`<rdfs:domain rdf:resource="&movieOntology;Movie"/>`
	`<rdfs:range rdf:resource="&xsd;string"/>`
	`</owl:DatatypeProperty>`
	…

movieOntology-imdb dictionary	`<owl:Class rdf:about="&movieOntology;Movie">`
	`<owl:equivalentClass rdf:resource=”&imdb;Movie”/>`
	`</owl:Class>`
	
	`<owl:DatatypeProperty rdf:about="&movieontology;title">`
	`<owl:equivalentProperty rdf:resource=”&imdb;title”/>`
	`</owl:DatatypeProperty>`
	…

Imdb ontology	`<owl:Class rdf:about="&imdb;Movie"/>`
	
	`<owl:DatatypeProperty rdf:about="&imdb;title">`
	`<rdfs:domain rdf:resource="&imdb;Movie"/>`
	`<rdfs:range rdf:resource="&xsd;string"/>`
	`</owl:DatatypeProperty>`
	…

Dictionaries can also use the skos:narrower and skos:broader properties to connect ontology entities, in case they are not semantically equivalent.

4.5.3 Mechanisms for exploiting CDS dictionaries

4.5.3.1 Publishing

Consider the “Star Wars” Resource VDI, described in the previous example. The metadata in the VDI is based on the movieOntology. The walkthrough below shows how the user can use the CDS to create semantically equivalent metadata based on the imdb ontology.

Step	User action	System response	Semantic description
1	User passes the movieOntology-based semantic description to the CDS		<rdf:rdf> <rdf:description rdf:about=”VDI1”> <rdf:type rdf:resource=”&movieOntology;Movie”/> <movieOntology:title>Star Wars</movieOntology:title> </rdf:description> </rdf:rdf>
2		CDS returns the user the imdb-based equivalent rdf description	<rdf:rdf> <rdf:description rdf:about=”VDI1”> <rdf:type rdf:resource=”&imdb;Movie”/> <imdb:title>Star Wars</imdb:title> </rdf:description> </rdf:rdf>

4.5.3.2 Subscribing

The CDS can also be used to translate a SPARQL query from one ontology to another, as shown in the walkthrough below.

Step	User action	System response	Semantic query

1	User passes the imdb-based SPARQL query to the CDS		SELECT ?x
			WHERE
			{
			?x rdf:type imdb:Movie
			?x imdb:title “Star Wars”
			}

		CDS returns the user the movieOntology-based equivalent SPARQL query	SELECT ?x
			WHERE
			{
			?x rdf:type movieOntology:Movie
			?x movieOntology:title “Star Wars”
			}

4.5.3.3 Matching

On receiving the publication, the CDS is requested to expand the semantic description of the VDI of the movie using the movieOntology-imdb dictionary. As seen in the walkthrough below, this procedure creates new triples.

Step	User action	System response	Semantic description
1	User passes the movieOntology-based semantic description to the CDS for materialization		<rdf:rdf> <rdf:description rdf:about=”VDI1”> </rdf:description> </rdf:rdf>
2		CDS returns the user the materialized rdf description of the VDI	<rdf:rdf> <rdf:description rdf:about=”VDI1”> <rdf:type rdf:resource=”&imdb;Movie”/> <imdb:title>Star Wars</imdb:title> <rdf:type rdf:resource=”&movieOntology;Movie”/> <movieOntology:title>Star Wars</movieOntology:title> </rdf:description> </rdf:rdf>

After expansion, movieOntology-based and the imdb-based SPARQL queries will both find the match. This mechanism is explained in detail in the section 4.13.

4.6 Semantic Overlay

4.6.1 Semantic Foundations

CONVERGENCE is built upon a content-centric network, which is accessed through the CoNet engine and its APIs. Hence, applications are not aware of the locations where resources are stored. To access a resource they use the name under which it is advertised in CoNet (see description of CoNet TE in section CoNet TE). However, a name cannot convey the whole meaning of the resource, which is needed for discovering the resource using some (semantic) description.

The indexing and efficient retrieval of content by semantic metadata is the key feature of the CONVERGENCE **Semantic Overlay**.

Efficient semantic operations on metadata require a scheme for semantic categorization of resources [38]. The first element in the CONVERGENCE scheme is the so-called CONVERGENCE Core Ontology (CCO); a CONVERGENCE-wide hierarchical taxonomy of resource semantic types, implemented as an OWL ontology. The first level root concept in the ontology is the “Resource”; first level children are “Digital Resources”, “People”, “Real World Objects” and “Services”, corresponding respectively to the *Internet of Media,* the *Internet of People,* the *Internet of Things and* the *Internet of Services*. Figure 16 provides an abstract view of the CCO.

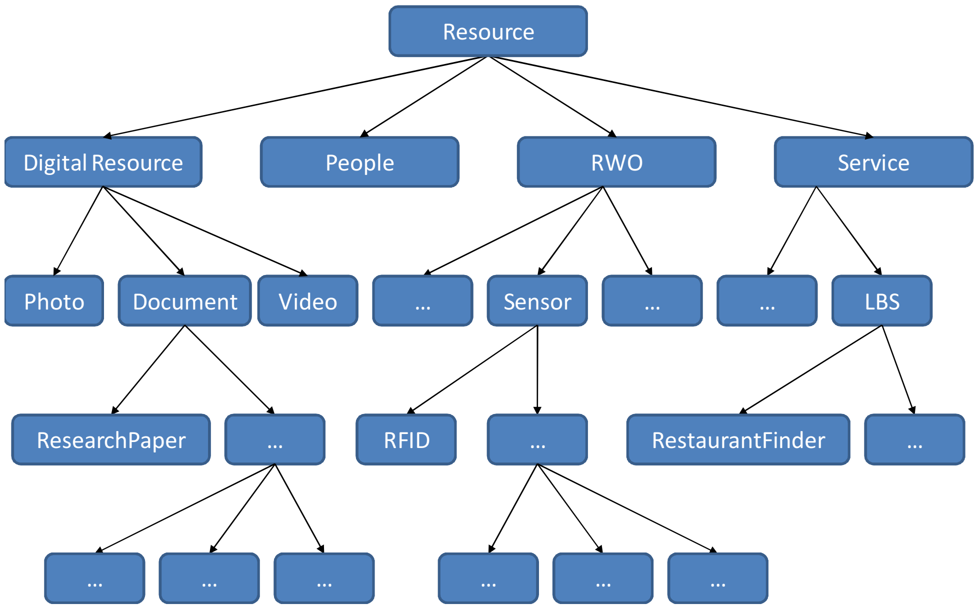


Figure 16 - CONVERGENCE Core Ontology (CCO)

The CCO is not the only ontology available to CONVERGENCE users. Additional ontologies make it possible to categorize content in other ways, which may either make it more specific (e.g. it could be an ontology of music genres or movie types) or they could refer to entirely different context (e.g. classifying the peers based on their location).

4.6.2 Semantically managing the Overlay Topology

Peers belonging to the overlay are partitioned into groups based on semantic criteria. For example, all peers injecting VDIs which belong to the category cco:movie, belong to the group cco:movie. Groups like this have the same kind of self-similarity we find in fractals, geometric shapes "that can be split into parts, each of which is (at least approximately) a reduced-size copy of the whole" [33]. We therefore call them *network fractals*.

A peer may belong to some fractals in the CCO and other fractals in another ontology – representing another *dimension* of semantic meaning (e.g. an ontology of access rights). This organization is *virtual* and not a physical one. Each peer propagates a message in one or more fractals, in one or more dimensions. Every time a peer decides to propagate a message to the overlay, it determines the final group of peers that should receive the message. These peers may belong to one or more fractals in one or more dimensions.

Consider a fractal *i* belonging to the *j*-th dimension and consisting of a group of peers, G_i^j. The message is then propagated to the peers of the group *G* that satisfy the condition:

$$G = \bigcap^{j} \bigcup_{i} G_i^j$$

This implies that:

- When a user wants to inject content into a set of fractals belonging to a single dimension, she sends this content to the peers of the union of these fractals.
- When a user wants to inject content into a set of fractals belonging to multiple dimensions, she takes the union of fractals for each dimension and injects the content into the intersection of these sets.

4.6.3 Propagation Protocol

A major challenge facing CONVERGENCE Semantic Overlay is the very dynamic nature of the CONVERGENCE system. Peers will continuously join and leave fractals for many different reasons:

- A peer is no longer interested in a topic (i.e. no publications or subscriptions for that topic are present in the peer) and leaves the fractal.
- A new peer is interested in a topic (makes a publication and/or subscription for that topic) and enters the fractal.
- A peer goes offline/online.
- A peer crashes.

CONVERGENCE will therefore employ a gossiping protocol, to propagate content in the fractal and to enable peers to maintain a partial view of their fractals, since gossiping protocols have been proven to be robust and scalable and, hence, a good solution for dynamic networks [34][35][37][36].

4.6.4 Peers Registration to Fractals

Each fractal maintains a registry, which is maintained and used by constituting peers belonging to the fractal to communicate with each other. The registry contains the following important information.

- The URI of the ontology used to partition the system into fractals. This information is used to access fractals that are higher or lower in the hierarchy than the current fractal.
- The fractal identifier. This shows which fractal the registry refers to.

- The size of the fractal (the number of peers in the fractal). This number is used to determine the size of this registry.

- Other characteristics of peers belonging to the fractal
    - The peer identifier.
    - The peer overlay propagation service endpoint. Publication and subscription VDIs are pushed to this peer via this service endpoint.
    - The date on which the peer is scheduled to leave the fractal (the latest expiration date for any publication or subscription a peer has injected into the fractal).

Peers remain in the registry until the leave date has passed. When a peer receives a VDI from a remote peer, it checks the VDI's expiry date and updates the leave date of the remote peer in the local registry accordingly. After a peer has selected the peers to which it will propagate content (*partial view*), it checks their leave dates. In case there are peers in the partial view whose leave dates have passed, the peer removes them from the partial view and the registry. Finally, it replaces the removed peers in the partial view with others from the registry.

Each peer periodically advertises a part of the registry under a given, standard, name, which is decided *a priori*. The selection of the name is system-dependent; for example, in CONVERGENCE, the registry for a fractal of type cco:movie, has the name `urn:overlay:registry:cco:movie`. This way, any peer that enters the system will always find another peer that can provide it with an entry point to the system. Since gossiping protocols do not require from a peer to communicate with all the other peers of the fractal, rather than with O(logN) of them, where N is the size of the fractal, the size of the advertised registry is of order O(logN).

Next, we are describing how the topology management protocol of the overlay handles the creation of a fractal and the update of the registry:

- **Bootstrapping.** When a peer enters a fractal (by issuing a publication or a subscription) it first has to register. To do this, it first requests the fractal registry by its name (as we said above, each peer periodically advertises a part of its registry). If there is no registry, this means that it is the first peer in the fractal. It therefore creates a registry, adds itself to the registry and advertises the registry to the network. If there is a registry it gossips a discovery message to the peers in the registry, which continue the gossiping of the discovery message. Each peer receiving a discovery message immediately adds the initiator (the first peer in the path contained in the header) to its registry, increases the fractal size by one peer and responds with its profile (identifier, overlay propagation service endpoint and leave date). When the new peer receives the profile, it adds it to the registry.

- **Registry Update (same ontology).** The bootstrapping procedure has the drawback that registries only contain entries for peers that are online at a given time. This is why every time a peer receives a gossiping message, it looks into the header, extracts the path and updates the registry with any peers that are not already in the registry. Every time a peer comes back online, peers that have been online in the meantime will have a better view of the system. The peer therefore checks the advertised registry of the fractal and adjusts the fractal size in its copy of the registry. Given the characteristics of the gossiping protocol, peers do not need to know the exact size of the fractal. The number of target peers and gossiping rounds (TTL) grows with logarithm of fractal size and do not change significantly with small changes in fractal size.

- **Registry Update (new ontology). I**n some cases, ontologies affecting the overlay may change. Such changes will affect the organization of the fractals and may lead to the addition of new fractals or the merging of existing ones. In the former case, the system needs to create a new registry for each new fractal. In the latter case, some fractals in the current view merge with the fractals in the updated view. In both cases, the system needs to process the core ontology and create semantic connections (equivalence links) between the old concepts and the new ones (see next section).If a peer realizes that the topology has changed (reflecting a change in the core ontology), it automatically converts the old resource semantic types to the new ones, using the connections defined in the new ontology. From that moment on, it gossips to the new fractal.

4.6.5 Message Propagation in Semantic Overlay

When a peer receives a message, it first checks its resource semantic type and *validates* it with the corresponding ontology (see Registry Update above). The peer then calculates the group *G* (see section 4.6.2) - its partial view of the system. To do this, it analyzes the relationships between the different fractals belonging to same dimension.

Consider the following example.

The VDI represents a publication about Action and Romantic Movies (suppose we have one fractal for each one of these genres) and also addresses peers that understand concepts about Hunting and Kisses (two fractals belonging to another dimension). The final group of peers will thus satisfy the relationship:

$$G = (Action \cup Romantic) \cap (Hunting \cup Kissing)$$ (See Figure 17).

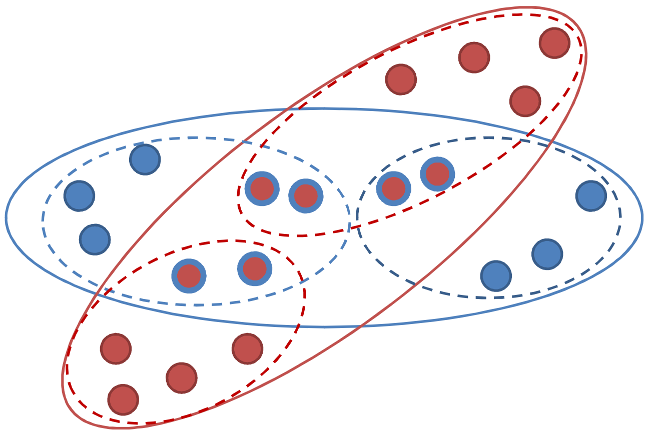


Figure 17–The target group of peers G will be consisted of the peers marked with both blue and red colour

Each peer then randomly selects log$|G|$ other peers from its partial view, sends them the Publication or the Subscription VDI and stops. This implies that peers do not send data at every gossiping round – only once every time they receive a Publication or a Subscription VDI. This procedure is repeated by each peer for *R* rounds, where:

$$R = \frac{\log|G|}{\log\log|G|}$$

Each peer that receives a Publication or a Subscription VDI during a gossip round, stores it in local buffers, which it will later be used for matching. On each round, the peer reads its buffers, reduces *R* by one, selects log$|G|$ peers uniformly at random and sends the VDIs to these peers using their publication or subscription service endpoints, given in the registry. The time between rounds depends on the trade-off we want between the delay of the VDI propagation and the resources' utilization.

Briefly:

- When a peer receives a Publication or a Subscription VDI, it stores it in its buffers. The VDIs that have not been gossiped by this peer yet are marked as ready to go
- When the time for gossiping has come, the peer reads its local buffers for VDIs marked as ready to go and:

- o determines the partial view *G*.
- o extracts peers to gossip to.
- o reduces the gossip message TTL (initially, TTL ≡ Rounds)
- o sends the gossip message (including the VDI) to the selected peers.

4.7 Event reporting

Event Reporting provides a standardized mechanism for defining, identifying and sending notifications about events, where an event is defined by a set of conditions which, when met, trigger an event catch. The main conditions involve Users, VDIs and specific type of action executed on VDIs. Major classes of "reportable" Events include the following:

(a) Events specific to the publish subscribe process, triggered by a "Match" action;
(b) Events generated within a peer during processing of a VDI by an application (e.g. play of a Movie VDI, t creation of an annotation for a VDI, etc).

The events listed under (a) only concern the process of content publication and the matching of it to subscription requests for concepts. For example, when a new VDI about a book is published, if some Users have subscribed to concepts related to the published content, a Matching event will occurred and an Event Report will be created to notify the users.

The events listed under (b) are associated with the domain logic of a particular application and are specific to it. This implies that before using Event Reporting for notifications, applications have to define and catch relevant events.

Event Reporting mechanism involves the two constructs defined below.

1. Event Report Request(ERR). An ERR defines the conditions that trigger an event. Events defined by an ERR trigger the creation of an ER which contains information describing the event, as specified in the associated ERR.
2. Event Reports(ER). ER are generated and sent according to the ERR. When the event occurs, an Event Report is generated and delivered to the specified recipient(s). The report contains the information requested.

4.8 Rights Expression Language

A Rights Expression Language (REL) is a machine-readable language that can declare rights and permissions using terms with an agreed semantics. A REL provides flexible, interoperable mechanisms to support publishing, distribution, and consumption of digital resources. This ensures that rights, conditions, and fees specified for digital resources are honoured and that personal data are processed in accordance with individual rights.
The REL standardized by MPEG [25] adopts a simple, extensible data model for many of its key concepts and elements (see Figure 18). The MPEG REL data model for a rights expression consists of four basic entities and the relationship among those entities. This basic relationship is defined by the MPEG REL assertion "grant". Structurally, an MPEG REL grant consists of the following:

1. The **principal** to whom the grant is issued
2. The **right** that the grant specifies
3. The **resource** to which the right in the grant applies
4. The **condition** that must be met before the right can be exercised

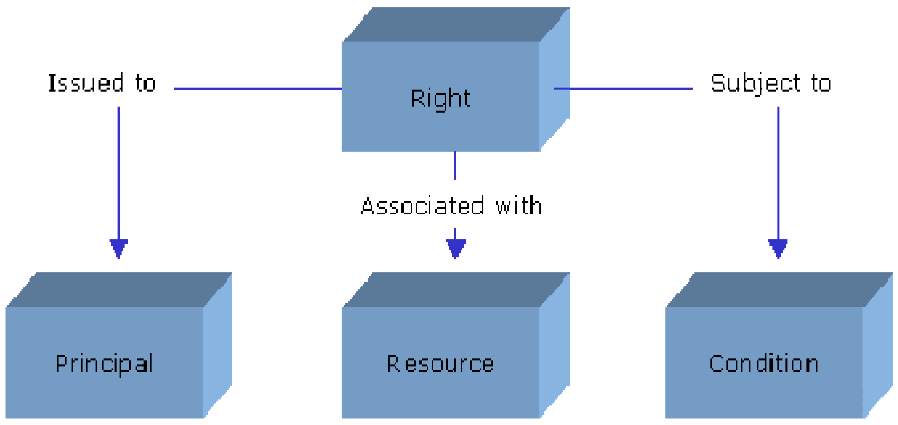


Figure 18 - The REL Data Model

Principal: encapsulates the identification of principals to whom rights are granted. A principal denotes the party that it identifies by information unique to that individual. Usefully, this is information that has some associated authentication mechanism by which the principal can prove its identity.
Right: the "verb" that a principal can be granted to exercise against some resource under some condition. Typically, a right specifies an action (or activity) or a class of actions that a principal may perform on or using the associated resource.
Resource: the "object" to which a principal can be granted a right. A resource can be a digital work (such as an e-book, an audio or video file, or an image), a service (such as an email service, or B2B transaction service), or even a piece of information that can be owned by a principal (such as a name or an email address).
Condition: the terms, conditions and obligations under which rights can be exercised. A simple condition is a time interval within which a right can be exercised. A slightly complicated condition is to require the existence of a valid, prerequisite right that has been issued to some principal. Using this mechanism, the eligibility to exercise one right can become dependent on the eligibility to exercise other rights. CONVERGENCE plans to use the existing language and semantics in Phase 1, but will substantially extend the elements of the language in the subsequent Phases to cope with an extended usage of the technology as demanded by CONVERGENCE.

4.9 Content identification

The CONVERGENCE system adopts three kinds of identifier: **VDI Identifiers, VDI Sequence Identifiers** and **Network Identifiers**.
VDI Identifiers are used by CONVERGENCE Middleware (CoMid) to uniquely identify VDIs.

The VDI Sequence Identifier is used by the CONVERGENCE Middleware (CoMid) to uniquely identify a sequence of VDIs so as to support information updating. The VDI Sequence Identifier is contained in a VDI.

When the information of a VDI needs to be updated, a new VDI is created with a different VDI Identifier, but with the same VDI Sequence Identifier. The VDI Sequence Identifier allows search and retrieval of the latest version of a VDI, or any preceding version. A VDI Sequence identifier has the form of a URI as better specified in D4.1 (section 6.2).

The Network Identifier is used by CoNet to identify named-data or a service access point. The named-data could be a VDI or a resource referred to in a VDI. A service access point is a network endpoint through which an upper-layer (CoMid or CoApp) entity sends/receives information.

From a user point of view, Network Identifiers are handled as URIs; “inside” CoNet they are transferred inside the CoNet data unit, as specified in D5.1 (section 5.4).

In what follow we describe how VDI identifiers are mapped to Network Identifiers in the default namespace. We use the following mapping:

VDI identifier:	urn:CONVERGENCE:eu:identify-content-server-id:label
Network Identifier:	identify-content-server-id/label

Where the identify-content-server-id is a unique identifier associated with the CoMid identify content server that released the VDI identifier and label is a unique identifier that differentiates the VDI identified by the server.

For instance,

VDI identifier:	urn:eu:CONVERGENCE:ics-CONVERGENCE:54ba64ed-a4e0-43e1-a22a-5e37a943ea19
Network Identifier:	ics-CONVERGENCE/54ba64ed-a4e0-43e1-a22a-5e37a943ea19

To identify the resource the VDI refers to (e.g. a file) we use a traditional domain-name/path structure directly mapped in a PLHB (Principal Label Hash Based) Network Identifier. For instance:

Plain domain-name/path:	test.wim.tv/CONVERGENCE-cedeo/ang.mp4
Network Identifier:	test.wim.tv/CONVERGENCE-cedeo/ang.mp4

where “test.wim.tv” is the identifier of the principal of the NID (e.g. the entity that can create NIDs using the *Principal Identifier*, see also section 7.1.3) and the label is “CONVERGENCE-cedeo/ang.mp4”

4.10 Publish / subscribe pattern

4.10.1 Description

Publish and Subscribe operations are carried out at the middleware level of the CONVERGENCE system. From the middleware perspective, publication of a Resource VDI involves the following main steps:

- Creation of a Publication VDI containing
    - Link to the stored Resource VDI (mandatory)
    - VDI Metadata, usually taken from the Resource VDI (mandatory)
    - Licence regulating access to the publication (optional)
    - Event Report Request defining reports to be issued when specific events occur (optional)
- Injection of the Publication VDI into the overlay of peers forming the "discovery overlay"
- Storage of the Publication VDI on the network

Symmetrically, the subscription process involves:

- Creation of a "Subscription VDI" containing
    - One or more representations of the semantic subscription to a set of metadata, in the form of a SPARQ query or a list of requested metadata (mandatory)
    - Licence regulating access to the subscription (optional)
    - Event Report Request defining reports to be issued when specific events occur (mandatory)
- Injection of the Subscription VDI into the overlay of peers forming the "discovery overlay"
- Storage of the Subscription VDI on the network

This procedure enables search for subscriptions using standard Search Content procedures (see D5.1 for the Search Content protocol API, see the following chapter for details of Semantic Search Mechanisms), and easy matching of publications to subscriptions. Users are thus given the possibility of revoking their subscriptions and publications, and of extending/altering them at a later time, as with any other VDI. By using VDIs to carry subscriptions and publications, we allow CONVERGENCE to exploit the Event Reporting mechanisms defined in MPEG-21 part 15, embedding Event Report Requests into each Subscription and Publication VDI. Subscription and Publication VDIs are uniquely identified, and carry the requested metadata plus a reference to the address of the "home" peer of the user to be notified if a match is found. The reference can take the form of a peer identifier or User VDI.

The use of specific licenses for Subscription VDIs and sequences of Subscription VDIs, enable many possible extensions. For instance licenses could restrict the kind of information that can be subscribed to by a particular class of users or support focused “forgetting of subscriptions”, so that some subscriptions immediately disappear from the system if not satisfied, while other persist indefinitely.

The subscribe operation is carried out by invoking a Subscribe Content Service. Similarly, the publish operation is carried out by a Publish Content Service. Both are complex operations, which involve a chain of Elementary Services. They are thus Aggregated Services. This implies there is no simple Publish Content ES, or Subscribe Content ES, but rather a pub/sub workflow involving coordination of middleware engines.

CONVERGENCE publication and subscription operations fully comply with the “content based” publish/subscribe paradigm as defined in [19] and support structured semantic descriptions of content. This is much better than “topic based” approaches that require subscribers to know the full name of the content they require.

Semantic subscription makes it possible to store requests for events that change the state of a certain specific VDI (e.g. creation of a new version of the VDI, revocation of the VDI, the creation of a link to the VDI). They also allow subscribers to subscribe to VDIs that have not yet been published e.g. VDIs for:

- New models produced by John;
- Special sales of mobile phones at a local store;
- Users want to know about all the associations made on columns of a certain author;
- Users want to be informed about releases of movies of their favourite actor;
- Users want to know how many times other users have done certain things on their content.

In CONVERGENCE the two cases above are equivalent and are treated using the same generic procedures.

When users create a subscription they issue a (semantic) query. The system returns a list of results. The user can then select the most relevant VDIs from the matching list, and/or wait for results that will be notified later. This requires a system wide request that is universally understandable. It also shows one of CONVERGENCE’S “Unique Selling Points”.

Today, when a user subscribes to a particular kind of CD in music.com (e.g. Iron Maiden CDs), she is informed of the CDs only when they are advertised in music.com. In CONVERGENCE her subscription is system wide and she receives a notification every time anyone publishes information about the CDs that interest her.

Conceptually, the subscription process is split into three parts:

- Part 1: inserting a semantic subscription system wide
- Part 2: matching a subscription once relevant content is published
- Part 3: delivering a notification to the subscriber

4.10.2 Part 1: inserting/storing a semantic subscription system wide

We have two users: P (who publishes) and S (who subscribes). Consider the case in which the subscription is prior to the publication of material matching the subscription.

When S subscribes, she provides semantic metadata, possibly organized in a complex query to filter out unwanted content and group and sort the results. These metadata are semantically validated by the CDS, which helps the user to formulate the query correctly (see relevant paragraphs about the CDS of this same deliverable, section CDS).

S may optionally provide a licence governing the use of the information contained in the Subscription VDI.

Along with these metadata, an Event Report Request is created. The ERR defines the event type that will trigger the notification (we call it a "Match" event type) and additional information needed to locate the peer that will receive the notification.

This ERR is inserted into the Subscription VDI. As mentioned earlier, it includes so called "DeliveryParams" (see[26]), that is the address of the peer to be notified if the event occurs. According to the Event Reporting standard this can be an URI, a CoNet network resource (or service) name, a reference to a User VDI, an email address, etc.

Therefore a Subscription VDI will contain:

1. A structured description of the metadata of the resource the subscriber is interested in
2. A Licence (defining who can do what with the subscription)
3. An Event Report Request containing
    a. The verb "Match" applied to this Subscription Metadata and any corresponding Publish Metadata
    b. The Identifier of the Device or User receiving the Event Report

This set of information is published to the system by injecting its content into the overlay and storing the full Subscription VDI on the network – the same mechanism used for publication.

4.10.3 Part 2: matching a subscription

Later on, when P publishes content, she also provides the (semantic) metadata that best describes the published resource. The metadata are then injected into the system.

When the gossiped information arrives at the destination peers, the procedure is reversed and all pending semantic search operations are performed on these peers, using those data as a target. If a subscription is found (say the subscription by S), that matches the target metadata, it becomes a match candidate. So, whenever a Publication VDI reaches a peer, at each publish, it is evaluated against outstanding semantic queries from Subscription VDIs known to the peer.

With this approach, we implement a sort of rendezvous node between Subscription and Publication VDIs in the peers responsible for the common Resource Semantic Type (see section 4.6).

4.10.4 Part 3: delivering matches to subscribers.

Each match between a new publication and existing subscriptions generates an Event Report - the companion of the ERR embedded inside the matched Subscription VDI. The ER is delivered to the peer/user referenced in the matched Subscription VDI, using an appropriate transport protocol (e.g. the CoNet SendTo primitive is invoked, or an email is sent). When the user receives the notification, (an Application working on her behalf) can fetch the matching VDIs directly using the Deliver Content protocol.

This solution makes use of the MPEG-21 Event Reporting standard, in conjunction with the concept of a Subscription VDI. In this way, we separate the MPEG-21 event notification mechanism from the internals of injecting metadata to peers. If required, the regular MPEG-21 event notification mechanism can still operate at the client-server level, within the scope of a custom application. To apply the MPEG-21 event notification mechanism system-wide we use specific Subscription VDIs to inject events. This requires the creation of a new verb (“Match”) in the Rights Data Dictionary (see[26]). Details of this extension to the MPEG-21 standard are left for future deliverables.

4.11 Digital Forgetting

4.11.1 Introduction

In today’s digital world, it is extremely easy for users to publish information via websites, blogs, video-sharing sites, social sites etc. Once the information has been published, it becomes instantly available to a potentially global audience. Search engines and web archiving tools ensure that copies of the information are rapidly disseminated beyond the site(s) where it was originally published. Private users may also make copies of and republish the information, usually without the knowledge of the original author.

As a result, it is practically impossible to locate all copies of information, and even harder to remove it from the network, *even when the author is able to remove the information from the site where it was originally published*. There are some circumstances in which this is what the author wishes. For instance, a political dissident may wish to ensure that it is impossible to eradicate her views from the network even if she herself should be forced to demand their removal.

However, there are many other circumstances in which a user may legitimately wish to eliminate information she has published. The most commonly cited reasons are personal e.g. the wish to eliminate references to opinions the user no longer holds; the wish to eliminate personal information that has become embarrassing for the user. In other cases, there may be business reasons to withdraw previously published information (e.g. a company's desire to update product information that has been shown to be inaccurate or information on promotional offers that are no longer valid). In summary, there is a mismatch between the Internet's tendency to preserve information forever and the needs of users.

4.11.2 Functional Requirements

Against the background described in the previous paragraph, one of CONVERGENCE's goals is to support the "Digital Forgetting" of content published on the CONVERGENCE network, allowing users to remove (or update, i.e. remove and re-insert a new VDI), information they have previously published. This goal is translated into formal requirements which are presented in section 7 of D2.1, (and will, therefore, not be readdressed here).

To satisfy these requirements effectively, it will also be necessary to meet a number of secondary requirements.

1	The creator of a VDI shall be able to define who can "unpublish" the VDI. The possible options shall include "no-one".
2	Any authorized user shall be able to unpublish the VDI
3	Derived VDIs shall, by default, inherit expiry dates and rights to unpublish from the original VDI, if the publisher does not provide new ones.
4	CONVERGENCE users shall not be able to retrieve an expired or unpublished VDI
5	VDI search and subscribe services shall not return references to expired or unpublished VDIs
6	Expired or unpublished VDIs shall be deleted from the CONVERGENCE network
7	VDIs stored off the CONVERGENCE network shall expose expiry date data to third party applications, allowing users to verify if they have expired (this supports the introduction of

	provisions to make use of expired VDIs illegal)
8	Users should be able to specify that a VDI has no expiry date (i.e. that it is intended to be “eternal”)

4.11.3 Digital Forgetting Mechanisms

To satisfy the requirements defined above, CONVERGENCE will support two forms of Digital Forgetting.

- Pre-planned Digital Forgetting – the CONVERGENCE system defines a default expiry date for all the Publication VDIs published to the CONVERGENCE system. The publisher of a VDI has the ability to override the default value at publication time. He/she can also define who (if anyone) is authorized to change the expiry date. This information is, by default, inherited by derived VDIs. Publication VDIs that have passed their expiry date are no longer retrievable by CONVERGENCE users and are no longer referenced in results from CONVERGENCE search and subscribe services. The associated Content VDI will also be eliminated from CONVERGENCE CoNet.

- On-demand Digital Forgetting – when publishing a Publication VDI, the publisher can define who (if anyone) has the right to unpublish it, (default value: the publisher). The CONVERGENCE system will enable authorized users, at any time after publication, to request the removal of a Publication VDI, (and associated Content VDI) from the system. The system will then proceed to asynchronously eliminate all stored versions of the Publication VDIs (and their associated Content VDIs), and eliminate all semantic metadata about the VDI, wherever it is stored in the system. Once this operation has been completed, the Publication VDI (and its associated Content VDI), will no longer be retrievable and will no longer appear in search or subscription results. The removal of the Publication VDI will not prevent access to, retrieval and consumption of other Publication VDIs in the same VDI sequence. Other Publication VDIs that declare relationships with the unpublished VDI will remain accessible. However, these relationships will then point to an irretrievable object. On reception of the revocation command, from CoMid, the CoNet performs the appropriate removal actions.

4.11.3.1 Pre-planned Digital Forgetting

During the publishing process, the publisher specifies the time and date at which the VDI will expire. The operational workflows will proceed as follows:

- At Pub VDI publication time:
    - The publisher specifies the date and time at which the Pub VDI will expire, which should be before the expiry date of the targeted Resource VDI (Res VDI).

    - At the CoMid level:
        - CoMid Publish Content Aggregated Service (PCAS) uses the Overlay TE to gossip the new Publication VDI through the system.
        - PCAS submits the Publication VDI to CoNet, for storage and distribution.
    - At the CoNet level:
        - CoNet stores and disseminates the submitted digital objects as sets of Named-data CoNet Information Units (CIUs, see 7.1) (with a field specifying their expiry date).
- At Publication VDI expiry time (more precisely, from the expiry time onwards):
    - At the CoMid level:
        - The Match TE which handles the matching between Pub and Sub VDIs, detects that the corresponding entry in the table is no longer valid and removes it.
    - At the CoNet level:
        - Serving nodes and border or internal nodes delete the named-data CIUs in question (those that contain the Pub and Res VDIs), and no longer cache or distribute them.

From the Publication VDI expiry time onwards, the CoMid level will no longer contain any trace of the expired Publication VDI and the Publication VDI will no longer be retrievable from the CoNet.

From the Resource VDI expiry time onwards, it will also no longer be retrievable from CoNet.

4.11.3.2 On-demand Digital Forgetting

In this mode of Digital Forgetting, an authorized user specifies his/her desire for the removal (unpublishing) of a Publish VDI for which he/she has “unpublish rights”.

Given that CONVERGENCE incorporates a dynamic set of peers that can join or leave the system at any time, complete and synchronous removal of a Publication VDI from CoMid is not possible. CONVERGENCE therefore adopts an asynchronous approach:

- An authorized user issues a request for the removal (unpublishing) of a Publication VDI.
    - At the CoMid level:

        - CoMid Unpublish Content Aggregated Service uses the Revoke Content PE (RCPE), requests relevant peers to eliminate the metadata contained in the Publication VDI to be revoked
            - The RCPE creates an UnPublish VDI which references a specific Pub VDI. The UnPub VDI declares that its targeted Pub VDI is no longer valid and should be eliminated from match tables.
            - The UnPublish VDI is gossiped throughout the peer collective (employing the Overlay TE) and ends up in the same peers as the corresponding Publish VDI.
            - On receiving the UnPub VDI, peers perform the appropriate "forgetting" activities.
        - The Unpublish Content Aggregated Service issues a revocation command to CoNet, for the deletion of the Pub VDI in question.
    - At the CoNet level:
        - On reception of the revocation command, from CoMid, the CoNet performs the appropriate removal actions.

Forgetting of the UnPub VDI is handled at the system level. The life cycle of these VDIs is defined as follows.

- At UnPub VDI production/injection time:
    - The UnPub VDI is mandatorily given the same expiry date as that of its associated Pub VDI.
    - at the CoMid:
        - The CoMid Revoke Content Aggregated Service (PCAS) uses the Overlay TE to gossip throughout the system, the new UnPub VDI.

          Given that the UnPub VDI has the same expiry date as that of its associated Publication VDI, it will circulate for as long as there is some possibility that metadata in the Pub VDI is still "circulating" in the system
- At UnPub VDI expiry time (more precisely, from the expiry time onward):
    - at the CoMid level:

- The Match TE detects that the corresponding entry in the table is no longer valid and removes it.

4.12 Real world descriptors

As a part of the CONVERGENCE vision, we need the ability to describe Real World Objects. However, creating descriptors for all type of objects would be a very major project. Many organizations have produced ontologies or XML schemas for specific domains. For instance, the Association for Retail Technology Standards (ARTS) of the National Retail Federation [42] has developed an XML based schema applicable across multiple retail vertical segments including general merchandise, grocery, convenience, food and drugs. This schema is useful and the CONVERGENCE definition of VDIs for the retailing scenario will “include” and take advantage of these efforts. However its main focus is on the exchange of item-related data between systems within the confines of a retail enterprise, and it is not perfectly adapted to the consumer electronics domain we will consider in the CONVERGENCE trials.

In CONVERGENCE we will develop additional domain ontologies and use them to generate suitable descriptors for VDIs about products in order to integrate them with existing retail applications.

For a standard coverage of this topic see ANNEX B – Survey of solutions of real world descriptors.

4.13 Semantic Search and Content Matching

This section provides a detailed explanation of the implementation of semantic search in CONVERGENCE. As in other P2P search overlay structures, the protocol is fundamentally asynchronous. However CONVERGENCE’S publish/subscribe paradigm provides additional decoupling in time and space. Search requests are carried in VDIs. An expiry date dictates whether the user wishes for an immediate reply (as in present-day search engines) or is willing to accept results which may come in the next hours or days. Aggregation of similar results is carried out by the event report service that notifies the user of relevant matches. Below we provide a detailed walkthrough of the process.

A CONVERGENCE-compliant Application is running on a random PeerX (maybe a public device in the city hall, or the user’s iPad). The application uses services and technologies offered by the CoMid.

0. User UserA, types in some metadata (e.g. resource MOVIE, and genres SCI_FI and CRIME), related to resources she wants to subscribe to.

1. The application invokes the CDS.

    - The CDS expands the request by finding appropriate concepts in known domain ontologies. For example, if the user accepts the IMDB Mapping Movie Ontology [41] (or if the search Application accepts it on her behalf) the CDS maps the user request to the imdb:Movie and imdb:Genre classes.

- The CDS formulates a correct semantic query that reflects the subscription. For example "SELECT ?x WHERE {?x a imdb:Movie . ?x imdb:genres "SCI-FI" . ?x imdb:genres "CRIME"}", where the prefix imdb: qualifies the IMDB ontology.

- The CDS helps the user find one (or more) Resource Semantic Types within the CONVERGENCE Core Ontology, and possibly within other semantic dimensions (see chapter on Overlay TE, section Overlay TE), appropriate for the subscription. For example the user might select the cco:VIDEO semantic type, where cco: is the prefix of the CONVERGENCE Core ontology. In this way, the user exploits the concept of fractals and focuses the search on peers that are interested in the cco:VIDEO type of content, (peers that have previously published or subscribed to cco:VIDEO content and have actively joined the fractal). The CDS prepares an incomplete metadata tag describing the Resource Semantic Type of the new Subscription VDI (the VDI identifier is still not known).

2. The application invokes Create Content.

   - A Subscription VDI is created which contains:

     - A unique identifier (SVDI_1)
     - The completed metadata indicating the core semantic type of the VDI, i.e. an RDF triple such as {SVDI_1 cco:hasReST cco:VIDEO}, embedded inside a didl:Descriptor tag.
     - The requested query, inserted into another didl:Descriptor tag
     - Licence and ERR
     - Expiry Date of the VDI.

   - The ERR says "notify peer PeerA if match". PeerA is the "home" peer of UserA. It is a peer that will receive the match notifications (the Event Reports).PeerA runs the Store Event and the Request Event servers "of the user".

3. The application invokes Inject Content (see Figure 19).

   - If PeerX is not yet a member of the cco:VIDEO fractal, it registers to it and joins.

   - Information about the Subscription VDI is circulated in the fractal. Thus, the Subscription VDI SVDI_1 reaches the peers that participate in the overlay fractal assigned to it, i.e. the fractal named "VIDEO" (a fully qualified name would be in the form `urn:overlay:registry:cco:video`).

   - Each peer extracts information contained within the Subscription VDI (the identifier of the VDI, the embedded query and any additional metadata, license, ERR) and copies it to its own "Subscriptions Table". The "Subscriptions Table" keeps track of all Subscription VDIs that reach the peer, and indexes them using their VDI identifier.

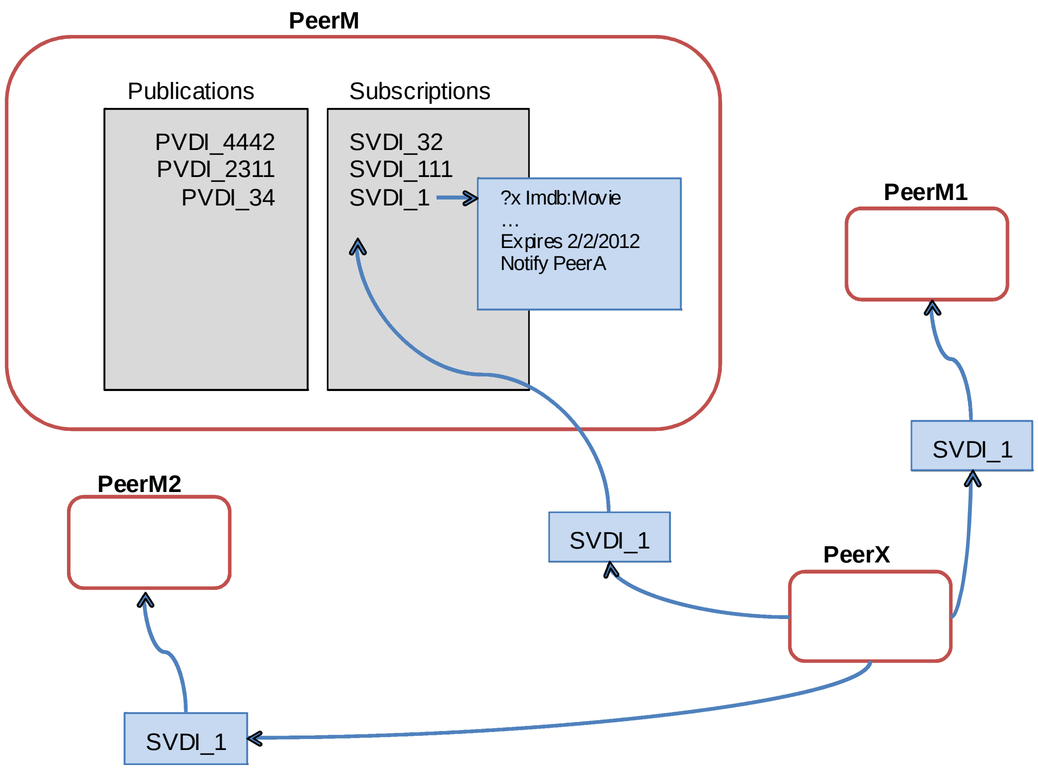


Figure 19 – Semantic Subscription SVDI_1 reaches PeerM, PeerM1 and PeerM2 of the VIDEO fractal

4. The application invokes Store Content.

   - The Subscription VDI is stored by CoNet as a generic network resource (just like any other VDI). This means that knowledge about the subscription is no longer restricted to the CoMid and can be **made available to crawlers that are not based on CoMid** (subject to security restrictions enforced by CoNet).

Another user makes a publication at a later time. That is: on a random PeerY (see Figure 20):

0. A user decides to publish a resource (e.g. a movie) for which he (or somebody else) has already created a Resource VDI. The Resource VDI describes the movie and the license that regulates access to it. The Publication VDI copies those Descriptors and adds an expiry date and license. Metadata describing the resource is collected from its Resource VDI, and possibly refined by the user. The user now has the role of a publisher. We assume the author of the resource has given her permission to publish it.

   - RDF triples describing the movie are extracted from its Resource VDI RVDI_23. The movie has been described using an alternative domain ontology about movies (e.g. the Movie

Ontology at www.movieontology.org), which focuses on a specialized taxonomy of movie genres).

- The following RDF Descriptor tags, taken from the original Resource VDI, are inserted into the new Publication VDI:
    - {RVDI_23 rdf:type movientology:Movie}
    - {RVDI_23 movientology:belongsToGenre movientology:Sci-Fi}
    - {RVDI_23 movientology:belongsToGenre movientology:Thrilling}
    - {RVDI_23 movientology:belongsToGenre movientology:Actionreach}

  The subject of the above semantic relationships is obviously the original resource VDI. They are now inserted into another VDI. This possibility is ensured by CONVERGENCE's concept of semantic links between VDIs. A specific tag allows RDF fragments to describe a VDI. In the case of publications, such VDIs simply state that they carry information about other VDIs.

1. The application invokes the CDS.
    - The CDS helps the user find one (or more) Resource Semantic Types within the CONVERGENCE Core Ontology or other shared semantic dimensions that provide appropriate classifications for the publication. For example the user selects the cco:VIDEO semantic type
2. The application invokes Create Content.
    - A Publication VDI is created that contains:
        - A unique identifier(PVDI_1)
        - The original didl:Descriptor metadata tags, now "describing" the Publication VDI
        - An RDF metadata triple describing the core semantic type of the Publication VDI and thus indicating the Publication VDI's destination fractal in the overlay (this information is embedded in another didl:Descriptor)
        - An explicit reference to the Resource VDI the Publication VDI refers to, i.e. {PVDI_1 cco:isPublicationOfRVDI_23}
        - Optionally a licence and an ERR, if the publisher wants to limit discoverability of the publication or to be notified when the publication is matched
        - Expiry Date of the VDI.
3. The application invokes Inject Content
    - If PeerY is not yet part of the cco:VIDEO fractal, it registers to it and joins

- Information about the Publication VDI is circulated: PVDI_1 reaches the peers that participate in the overlay fractal assigned to it, i.e. the “VIDEO” fractal.

- Each peer extracts information contained in the Publication VDI (the identifier of the VDI, all metadata, license, ERR) and copies it to its own “Publications” table. The “Publications” table keeps track of all Publication VDIs that reach the peer, and indexes them using their VDI identifiers.

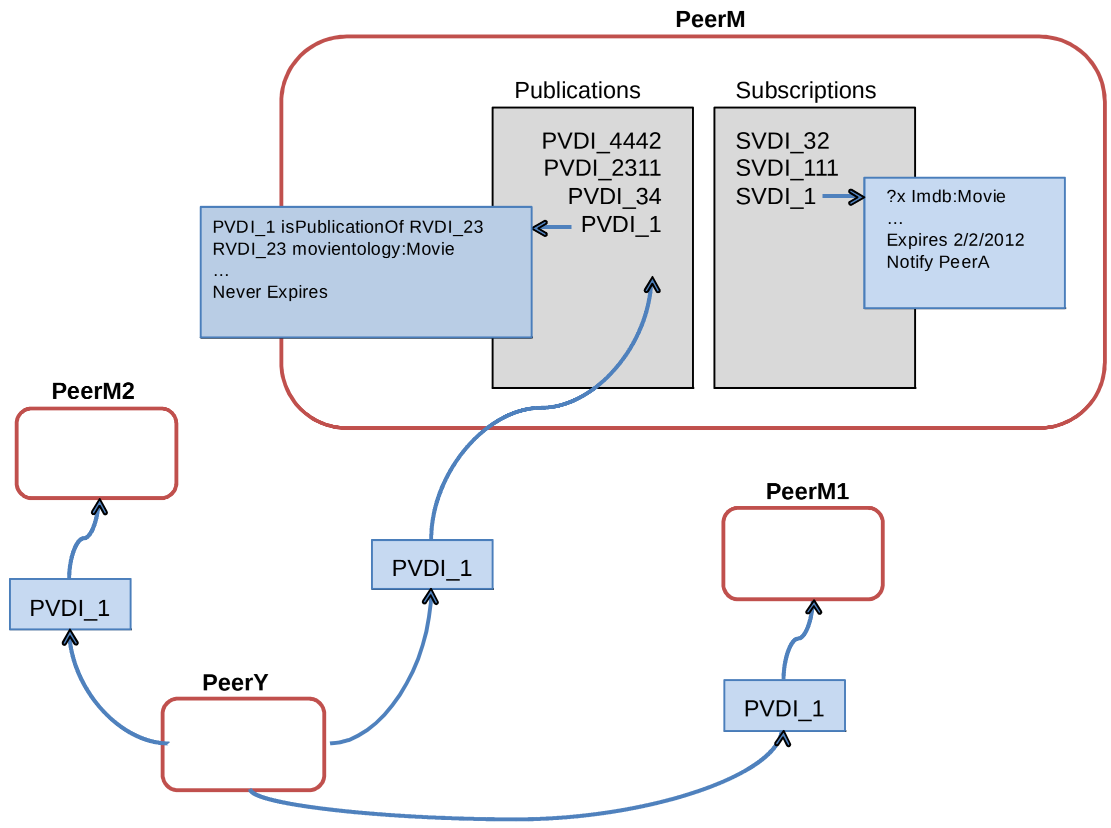


Figure 20 – Publication PVDI_1 reaches PeerM, PeerM1 and PeerM2 of the VIDEO fractal

4. The application invokes Store Content.

    - The Subscription VDI is stored by CoNet as a generic network resource generic network resource (just like any other VDI). This means that knowledge about the subscription is no longer restricted to the CoMid and can be made available to crawlers that are not based on CoMid (subject to security restrictions enforced by CoNet)

The publication and the subscription VDIs reach several rendez-vous peers of the VIDEO fractal. Let us observe one of them. On PeerM (see Figure 21):

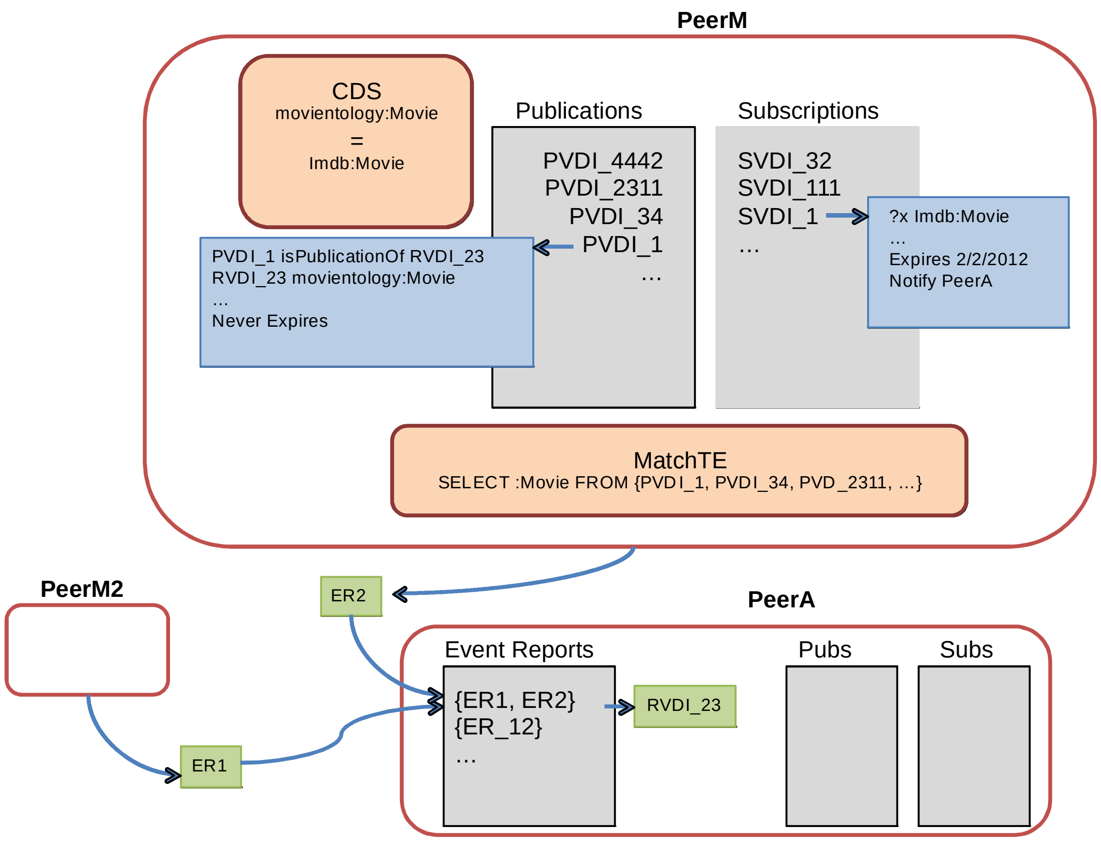


Figure 21 – Subscription is matched by PeerM and PeerM1

5. Match TE constantly monitors peer tables and is triggered on arrival of a Publication VDI or a Subscription VDI. The goal of the Match TE is to perform a match between the SPARQL queries embedded inside the Subscription VDIs stored in the "Subscriptions" table, and the metadata stored in the peer's "Publications table".

    - When a Publication VDI arrives at the peer, the full list of SPARQL queries in the "Subscriptions" table is scanned, and run against the newly arrived metadata.

    - When a Subscription VDI arrives at the peer, the embedded SPARQL query is run against the full list of metadata found in the "Publications table".

    Before attempting to match a SPARQL query against the available data, the system evaluates possible semantic equivalences between concepts belonging to different ontologies. To do this

    - Match TE invokes CDS.

- The CDS asserts a semantic equivalences between the imdb:Movie **class** and the movieontology:Movie **class,** between the movientology:belongsToGenre **property** and the imdb:genres **property,** and between the movientology:Sci-Fi **individual** of type movientology:Genre and the “Sci-Fi” **literal** assigned to the imdb:genres property[3]. The task of finding equivalent genres is greatly facilitated by the preliminary assertion of the equivalence of associated properties (preceding bullet), and of course matching of the literal with the individual name.

If more than one SPARQL query is to be executed the Match TE runs a query engine capable of processing semantic queries. The target data for the query is the aggregate of all metadata relevant to the query, after query expansion of the semantic classes. Match TE asks what are the RDF triples that satisfy the query and fetches them.

If matching RDF triples are found, a "Match Event" is triggered. The ERR is extracted from the Subscription VDI, the address of PeerA is read and companion ER is generated.

The ER is filled with the identifiers of all Resource VDIs known to PeerM, which contain matching metadata RDF triples.

6. Match TE invokes Event Report TE

   - The ER is sent to PeerA using the transport protocol specified in the original ERR, and invoking PeerA’s Store Event service.

Different peers in the VIDEO fractal send Event Reports, notifying successful match events, and deliver them to PeerA. Note that some of the peers of the fractal may not be able to infer semantic equivalences because not all CDS engines have the proper dictionary to translate between the movieontology: and imdb: ontologies.

0. Since PeerA may receive several match notifications from different peers of the fractal, the peer performs a “notification fusion” eliminating duplicate VDI ids matching the subscription.

   When the user wants to check whether her home peer has received new notifications, she polls the Request Event service of PeerA from PeerX, where she is currently located. Alternatively, if PeerA is the user’s mobile/laptop of the user, a GUI alert pops when the ER is received.

[3]The IMDB ontology contains a detailed taxonomy of genres in the form of a class hierarchy. The Movieontology allows representation of genres through string literals.

4.14 CoMid Security

This section details the methodology introduced in section 3.4, starting with the identification and authentication of CONVERGENCE entities.

4.14.1 Roles

Among other things, the CoMid security provides the technical means for handling identification and authentication of CONVERGENCE entities. In this section, we provide the definition of these concepts, we identify the roles responsible for implementing these concepts, the assets that are the objects of these security concepts, the threats related to these assets, the objectives needed to address the threats and the security functional requirements that derive from the previous objectives. We will begin by defining identification and authentication:

Table 1 – Definition of Identification and Authentication in CONVERGENCE

Term	**Definition**
Identification	This is the process of assigning an identifier to an entity (like a user, end-user, device, etc.). Identification is usually considered to be a sub process of registration, and is essentially performed once for each entity prior to any action performed by him or it.
Authentication	This is the process of proving a claimant's identity on the moment of request. Authentication is a regular procedure usually required prior to processing any request of a claimant.

Note: The first definition is taken from the MPEG environment and deviates from common definitions for “identification”, which often comprise authentication (proof of identity) or determination of an identity.

The two basic roles with regard to the handling of identification and authentication in CONVERGENCE scenarios are **Identity Provider** and **Service Provider**.

Identity Provider

The Identity Provider is responsible for the registration of users. This may involve:

1. Identification, i.e. assignment of identifiers; including issuance of certificates
2. Verification of credentials
3. Enforcement of governing policies

4. Revocation/Renewal of identities

Depending on the scenario, registration can be a simple assignment of an identifier from a name space, or an extensive procedure involving validation of personal data and additional information.

The Identity Provider is a trusted third party usually independent of the users within a specific scenario. To mention some examples, in a university scenario the Identity Provider might be a commission made up of professors, while in a governmental application it might be a government agency.

The Identity Provider does not normally interfere with the "regular" daily business of using the identifiers assigned to entities.

Service Provider

The Service Provider is in charge of "daily" business involving users, entities, etc. for a specific scenario. Apart from its general obligations (depending largely on the underlying application) these may comprise:

1. Authentication, i.e. verification of claimant's identities, and proof of its own identity.
2. License issuance
3. License validation

Although the two roles – Identity Provider and Service Provider – may coincide for special applications, they are normally strictly separated. On the one hand, an Identity Provider may seek private information from users which are not intended to be revealed to Service Providers. (The Service Provider needs to trust the Identity Provider for correct verification.) On the other hand, detailed data gathered by Service Providers should not be disclosed to an Identity Provider. (For instance, detailed profiles of customers' shopping behaviour should remain confidential to a Service Provider; especially in the case of different Service Providers who are competitors in the market.)

4.14.2 Assets

Assets basically comprise:

a. Assets related to VDIs
b. Assets related to users (end-users, peers, etc.) and services (service applications)
c. Assets related to engines (software packages) in the CONVERGENCE middleware.

The following assets have been identified for the CONVERGENCE framework:

Table 2 – CoMid security Assets

No	ID	Asset	Description

1	ASS.VDI-ID	VDI Identification	Each VDI is assigned a unique identifier. A VDI identifier is never used twice, not even for two related or “updated” VDIs.
2	ASS.VDI-INT	VDI Integrity	A VDI comprises components such as content, metadata, annotations, inter alia. The VDI's unique identifier is also considered one of its components. The integrity of a VDI therefore refers to the VDI as a whole, including its identifier. A VDI should always and forever remain unchanged once it has been published, until possible revocation. The identifier must be unique for each VDI, in particular the identifier “Anonymous” is not allowed.
3	ASS.VDI-AUTH	VDI Authenticity	Beyond integrity, VDI authenticity refers to the origin of a VDI. VDI authenticity describes the assurance for a user that the VDI has indeed been created by the entity claiming its creation.
4	ASS.USER-ID	User Identification	A user may be a human end-user, or a device, or a network peer. Like a VDI, a user is identified by assigning an identifier to him or it. Identifiers are unique in the sense that no two users are allowed to share the same identifier. Unlike the situation for VDIs, one and the same user may – under certain circumstances – hold different identifiers at the same time. Even the identifier “Anonymous” shall be supported, depending on the underlying policy.
5	ASS.USER-AUTH	User Authenticity	Users authenticity refers to assurance of the “integrity” of a user at the moment of his or its identity claim. In other words, the authenticity of a user claiming a specific identity describes the fact that the claimant is indeed the one to whom the identity has originally been assigned.
6	ASS.SERV-AUTH	Service/Application Authenticity	Service (Application) authenticity defines the integrity of the service upon the moment of its

			identity claim. Each user (in particular human end-user) shall – at his or her discretion – gain assurance that a service claiming a specific identity is indeed the one to which this identity has been assigned.
7	ASS.ENG-ID	Engine Identification	This asset refers to the identification of an engine.
8	ASS.ENG-AUTH	Engine Authentication	This asset refers to the authenticity of an engine, here meaning its integrity and possibly origin.
9	ASS.ENG-VDI	VDI technology engines	This asset refers to technology engines handling VDIs.
10	ASS.TE-LIC	License Technology Engine	This asset refers to licensing technology, referring both to issuance and compliance.
11	ASS.TE-ER	Event Reporting Technology Engine	This asset refers to monitoring and reporting of events.
12	ASS.SEC	Security Technology Engine	This asset refers to software and hardware handling cryptographic services, and interfacing CoSec.

4.14.3 Threats

The following threats have been identified for the CONVERGENCE framework:

Table 3 – CoMid Security Threats

No	ID	Description
1	T.VDI-ID-NONE	A VDI may not have been assigned an identifier prior to publishing.
2	T.VDI-ID-MULT	One and the same VDI may be assigned multiple identifiers.
3	T.VDI-ID-NONUNIQUE	A VDI may be assigned an identifier which has already been assigned to another VDI.
4	T.VDI-ALTERED	Components of a VDI may be altered after publishing.

5	T.VDI-ID-REPLACE	The identifier of a VDI may be exchanged with another identifier; the remaining content remaining unchanged.
6	T.VDI-CONTENT-REPLACE	The (entire) content of a VDI may be replaced or altered while keeping its identifier. **Note**: T.VID-ID-REPLACE and T.VDI-CONTENT-REPLACE are special cases of T.VDI-ALTERED. The identifier of a VDI is considered to be a component of the VDI.
7	T.VDI-AUTH-REMOVE	The origin of a VDI which has originally been stated may be removed, making it impossible to determine its creator.
8	T.VDI-AUTH-ALTER	The origin of a VDI may be altered, suggesting that the VDI has been created by an entity different from the one that really created it.
9	T.USER-ID-NONE	A user does not receive an identifier although – by the policies in place – he or she is required to hold one.
10	T.USER-ID_MULT	A user may be assigned multiple identifiers in a situation where unambiguous identification is a requirement.
11	T.USER-ID-AMBIG	A user may be assigned an identity which has already been assigned to another user.
12	T.USER-AUTH-FORGE	A user tries to authenticate himself or itself under a “false” identity which has not been assigned to him (i.e. it has been assigned to someone else, or not assigned to anyone at all).
13	T.SERV-AUTH-FORGE	A service (application) tries to authenticate itself under a “false” identity, namely an identity which has not been assigned to it.
14	T.SERV-AUTH-MISS	A service fails to authenticate itself although it has explicitly been requested to do so by another user, or is implicitly required to do so under policies in force.

4.14.4 Objectives

The following security objectives have been identified for the CONVERGENCE framework:

Table 4 – CoMid Security Objectives

No	ID	Description
1	O.VDI-ID-UNIQUE	In order for each VDI to obtain a unique identifier, it is not up to the discretion of the creator of a VDI to assign an identifier to it.

		Instead, servers will draw these identifiers from disjoint name spaces and assign an identifier to each VDI before publishing. Following publishing, no VDI may be altered in any way prior to possible revocation.
2	O.VDI-INT	The integrity of a VDI is regarded as critically important: A VDI must remain unaltered throughout its lifetime, i.e. after publishing and before revocation. This refers to its entire extent, including in particular its identifier, and any components like metadata, references, annotations, or content.
3	O.VDI-AUTH	Where required or granted by the creator, the origin of a VDI must remain unchanged throughout its lifetime. In such cases, a user shall be able to gain assurance about the authenticity of a VDI. **Note**: however that it may be allowed – depending on the policies in place – to create a VDI anonymously, or on behalf of a group of users, or under a pseudonym.
4	O.VDI-AUTH-ALTER	It shall be impossible – or at least detectable – to alter the origin of a VDI that has been stated during its creation. To this end, if an origin is claimed, the origin shall be ascertained during creation in such a way that it can be verified by an independent party.
5	O.VDI-AUTH-REMOVE	It shall be impossible – or at least detectable – to remove a claim (and assurance) of origin of a VDI which has been included by its legitimate creator.
6	O.USER-ID	A service shall be provided responsible for assigning a unique identifier to each user. This process shall be part of a registration procedure, governed by an independent party acting in the role of an Identity Provider. **Note**: The interpretation of the registration procedure may vary considerably for different scenarios, ranging from simple identifier assignment from a name space to governmental procedures encompassing passport or even biometric validation.
7	O.USER-AUTH	Each user shall be able to authenticate his claimed identity to an independent party. Note however that we allow four types (categories) of identities a user may hold. All four shall be generally supported by CONVERGENCE. The identities actually supported depend on specific use cases. The four types of identities will be outlined in the section “Security Functional Requirements” as F.USER-ID-SINGLE, F.USER-ID-GROUP, F.USER-ID-PSEUDONYM and F.USER-ID-

		ANONYMOUS.
8	O.SERV-ID	Unlike for users, services (applications) shall have a unique (non-anonymous) identification.
9	O.SERV-AUTH	A server shall be able to authenticate to a user at any time. A server shall authenticate to a user at the user's discretion, or if required by underlying policies.

4.14.5 Security Functional Requirements

The following security functional requirements have been identified for the CONVERGENCE framework:

Table 5 – CoMid Security Functional Requirements

No	**ID**	**Description**
1	F.USER-ID-UNAMBIG	The Identity Provider shall ensure that no identifier is used twice for different users. (This requirement does not conflict with the group identity, since in such case the group as a whole is considered as the user in the sense of the definition above.) **Note**: It may however – depending on policies – be admissible to assign multiple identities to one and the same user.
2	F.USER-ID-UNIQUE	If required by underlying policies, the Identity Provider shall be able to ensure that each user will receive a unique identifier.
3	F.USER-ID-MULT	Depending of underlying policies, a user may have multiple identities: a. In the *CONVERGENCE instance,* if it so allows b. With different Service Providers, if they so allow c. With the same Service Provider, if he so allows
4	F.USER-ID	A user shall be identified by an Identity Provider in order to be able to authenticate and access specific services of a CONVERGENCE instance. Moreover, a user shall be offered the possibly Service Provider - dependent choice of identification listed below as F.USER-ID-SINGLE, F.USER-ID-GROUP, F.USER-ID-PSEUDONYM and F.USER-ID-ANONYMOUS. a. An individual/organization or b. A member of a group without disclosing his individual identity (identity can only be revealed by the group

		administrator) c. A context dependent Pseudo-identity d. Entirely anonymous
5	F.USER-ID-SINGLE	A user (end-user as well as organization) shall be able to authenticate himself under a single – and unique – individual identity confirmed to him by an Identity Provider. Note that this type is considered to coincide with the "ordinary" use of an identity to be normally confirmed through a proof of possession of a specific secret. Note also that this does not rule out the use of multiple single identities being assigned to one and the same user where policies admit this.
6	F.USER-ID-GROUP	A user shall be able to register as a member of a specific group of users. This registration is carried out by the Identity Provider setting up this group. In this case, the user shall be able to authenticate himself as a legitimate member of the registered group, though remaining entirely anonymous within the group. *This type of identity is supported by cryptographic procedures known as group signature schemes. These schemes allow members of a group to issue signatures on behalf of the entire group, while staying anonymous as individual members of the group. The schemes also allow unveiling of anonymity in the aftermaths, and revocation of individual members with the explicit cooperation of the "group administrator". In other words there is an entity which is able to unveil anonymity, as it may be required by regulation.*
7	F.USER-ID-PSEUDONYM	Each user shall be able to hold context-specific pseudonyms, and to authenticate himself within each context by the according pseudonym. A user's pseudonyms are unique within each context, but different and not linkable across different contexts. Pseudonyms are generated according to a key scheme set up, and not at the discretion of the users. This type of pseudonyms is supported by the technique of "Restricted Identification" as deployed by the German Electronic Passport.
8	F.USER-ID-ANONYMOUS	A user shall be able to act under an entirely anonymous identity, making his appearances absolutely indistinguishable from any other instance of other users of even his own. In other words, the identity "Anonymous" shall be considered as "void".

9	F.USER-AUTH-CONV	A user shall be authenticated in order to access specific services of a CONVERGENCE instance, using one of the four types of identities listed under F.USER-ID.
10	F.USER-AUTH-SERV	Users may need to be further authenticated by a service application, depending on specific requirements of the application.
11	F.SERV-ID	Each application service shall be identified by one or more Identity Providers with a trust relationship with the entity running the CONVERGENCE instance. Each application shall receive a unique (non-anonymous) identifier.
12	F.SERV-AUTH	A user shall be able to request authentication of an application. An application shall be able to authenticate upon such request, and is required to do so depending on policies.
13	F.PEER-AUTH	Peers may need to mutually authenticate.
14	F.ENG-ID	Identification is a CONVERGENCE-governance issue and ensures that any identifier is unique. Engines shall be identified by an Identity Provider.
15	F.ENG-INT	Engines shall have the ability to verify (i.e. check the integrity of) an engine.
16	F.ENG-AUTH	An engine shall be authenticated by a peer downloading and installing it. **Note**: Assuring correct download and installation is not enough to guarantee proper operation throughout the lifetime of the device. However, since device integrity is a far too complex matter to be handled by CONVERGENCE, we make the assumption that – once correctly installed – engine software will be operated on a trustworthy device. The usage of smart card technology can to some extend mitigate this assumption, since at least a partial control of device integrity becomes possible when using the smart card as a trusted hardware security module.
17	F.ENG-VDI	The VDI TE shall have the ability to: a. *Identify* a VDI and its components b. *Sign* a VDI and its components (*individual* and *group signature*) c. Make accessible parts of a VDI only to certain users (*access restriction*)

		d. *Authenticate* the creator of a VDI e. *Verify* a VDI and its components (verify the integrity) **Note**: c. will usually require encryption. The challenge arising in a context where many (or most) VDIs are encrypted for many recipients does not lie with encryption itself, but with key escrow. In other words, efficient handling of key management will become a major concern for system administrators trying to build up a CONVERGENCE instance using VDIs with access restriction.
18	F.ENG-VDI-SIGN-THPART	A user may use the services of a trusted third part to sign a VDI. **Note**: this requirement reflects a privacy concern arising in an environment where end-users regularly sign their VDIs. An abundance of signatures may lead to serious privacy concerns since they allow tracking and profiling of users (even for encrypted VDIs).
19	F.TE-LIC-ISS	The License TE shall have the ability to create a license whose principal may be a user as defined above.
20	F.TE-LIC-ENF	The License TE shall have the ability to enforce a license, i.e. force a peer to comply with rights included in a license.
21	F.TE-ER	The Event Reporting TE shall have the ability to force a peer to issue a message to the users and peers specified in the Event Report Request (ER-R) whenever the peer executes one of the verbs specified in the ER-R.
22	F.SEC	The Security TE shall have the ability to a. Encrypt/decrypt i. A resource using a *symmetric* key ii. A key using an *asymmetric* key (by CoSec) b. Create new *credentials* and manage *certificates* c. Generate *symmetric* keys Store confidential information e.g. licenses and keys in the secure repository Moreover, it shall interface CoSec to achieve a. Generation of asymmetric key pairs b. Management of group signature scheme c. Management of Identity Based Encryption Schemes d. Management of Attribute Based Encryption Schemes e. Management of pseudonyms

4.15 CoNet Security

In this section we provide a preliminary description of security and privacy issues that are of concern to CoNet. This section only describes security/privacy assets, threats and objectives. Functional requirements will be included in the next deliverable. As far as concerns CoNet security, we plan to implement functionalities that make it possible to accomplish the security objectives. As far as concerns CoNet privacy, we currently plan only to identify privacy objectives and to devise a CoNet architecture that does not prevent the implementation of the necessary privacy preserving mechanisms.

We stress that CoNet security and privacy mechanisms are completely decoupled from the mechanisms adopted at the middleware level (CoMid).

From a high-level point of view, the CoNet security and privacy assets to protect are, respectively:

- *Service integrity*, i.e. access to valid named-resources
- *Producer/user anonymity*

in cases where:

- Infrastructure may not trust users;
- Users may not trust infrastructure;
- Nodes of infrastructure may not trust each other.

We observe that, at least currently, we do not plan to use CoNet to support *confidentiality*. The rationale for this choice is on the one hand, that confidentiality is mostly an end-to-end issue, on the other, that CoMid already provides *confidentiality* support. The final decision on this issue will be described in future deliverables.

In the following sub-sections we describe the security and privacy assets, threats and objectives summarized in the table below.

Table 6 – CoNet security and privacy assets, threats and objectives

(security asset 1.1)	Integrity of name-system routing entries
(security asset 1.2)	Integrity of the name-system responses
(security asset 2.1)	Integrity of named-data
(security asset 2.2)	Originality of named-data
(privacy asset 1.1)	Anonymity of the producer of named-data
(privacy asset 1.2)	Anonymity of a user that requests named-data
(security threat 1.1)	An adversary has the possibility of remote accessing the routing database of the name-system-node
(security threat 1.2)	An adversary may intercept and then forward packets of a

	communication occurring between a node (border and end-node) and the name-system-node
(security threat 2.1)	An adversary may generate fake version of named-data
(security threat 2.2)	An adversary may intercept, alter and then forward named-data
(privacy threat 1.1)	The behaviour of a user/producer may be monitored and profiled by overhearing CoNet packets. The overhearing could occur in any part of the network, with the exclusion of the local access network (e.g., ADSL link, WLAN, etc.), which is trusted by the user
(security objective 1.1)	CoNet shall protect the routing database of the name-system from unauthorized modifications coming from remote users, i.e. Insert, modify and delete entries
(security objective 1.2)	CoNet shall avoid the name-system impersonification and the altering of name-system responses
(security objective 2.1)	CoNet shall avoid the caching of fake version of named-data
(privacy objective 1.1)	CoNet should not prevent to implement mechanisms that make possible to verify integrity and originality of named-data without disclosing the identity of the user that has produced the named-data
(privacy objective 1.2)	CoNet should not prevent the implementation of mechanisms that make it possible to deliver named-data without disclosing to intermediate un-trusted nodes the identity or network location of the requesting user

4.15.1 CoNet Service Integrity

In a content-centric network, access to a named-resource could be jeopardized by several kinds of malicious attacks, targeted at different networking mechanisms. In order to identify the possible security threats we adopt an architecture-centric approach, looking for types of attacks against the functionalities of the CoNet architecture. Obviously, the security threat model will be extended appropriately during the project lifetime.

4.15.1.1 Routing-by-name

CoNet routing-by-name is mainly carried out by means of the *lookup-and-cache* approach described in D5.1 and in this deliverable (see 7.1). Briefly, the name-based routing table of end-nodes and border-nodes is used as a *routing-cache*. If a routing entry is missing, the node lookups the routing entry on a name-system.

To preserve the routing-by-name mechanism, the security assets we need to protect are:

- (security asset 1.1) Integrity of name-system routing entries
- (security asset 1.2) Integrity of the name-system responses

We assume that an adversary may:

- (security threat 1.1) have the possibility of remote accessing the routing database of the name-system
- (security threat 1.2) intercept and then forward packets of a communication occurring between a node (border and end-node) and the name-system

We further assume that remote access to the name-system is always guaranteed, i.e. that there are mechanisms in place to prevent DoS attacks, for instance the saturation of the name-system with a huge number of requests.

To preserve assets against these threats, CoNet security objectives shall include the ability to:

- (security objective 1.1) protect the routing database of the name-system from unauthorized modifications (insert, change, delete entries) by remote users,
- (security objective 1.2) avoid impersonification of the name-system and the altering of name-system responses

4.15.1.2 In-network Caching

A fundamental feature of the CoNet is the ability of network nodes to cache chunks of named-data, i.e. named-data CIUs. Any border or internal node can cache chunks of named-data that it is forwarding, and can replay a cached chunk of named-data on request. This possibility offers undeniable performance benefits. However, it creates the risk that fake versions of named-data could prevent the retrieval of the original version. A fake version of named-data is some named-data with the same name (NID) as the original named-data, but different content. If an intermediate node (border or internal) has unintentionally cached a fake version of named-data, it will respond to requests for the original named-data with the fake version, preventing end-nodes that make such requests from downloading the original data.

To preserve the possibility of retrieving the original version of named-data, the security assets we need to protect are:

- (security asset 2.1) Integrity of named-data; i.e. the certainty that named-data have not been modified since they were made available by the author
- (security asset 2.2) Originality of named-data; i.e. the certainty that named-data have been created by the original author, that is the user with the right to use the name (e.g. www.cnn.com) of the named-data.

An adversary may:

- (security threat 2.1) Generate a fake version of named-data
- (security threat 2.2) Intercept, alter and then forward original named-data

However, for the scope of this work, we assume that cached named-data cannot be manipulated, i.e. the cache storage unit is trusted.

In order to preserve in-network caching assets against these threats, CoNet security objectives shall include the ability to:

- (security objective 2.1) avoid caching of fake versions of named-data

4.15.2 CoNet Producer/User Privacy

CoNet content-centric networking paradigm poses new challenges for protecting user privacy. For instance, protecting the integrity and originality of cached named-data might lead to disclosure of the identity of the author. Similarly, the source-routing approach used by CoNet to deliver named-data to a requesting user might disclose the network location and thus the identity of the requesting user. This said, the privacy assets we should protect are:

- (privacy asset 1.1) the anonymity of the author of named-data
- (privacy asset 1.2) the anonymity of the user who requests the named-data

There are several possible threats to these assets:

- (privacy threat 1.1) an adversary could monitor and profile the behaviour of a user by sniffing CoNet packets.

At the time of writing, we assume that packet payload is encrypted at the CoMid level and is therefore not accessible to an adversary. However, this assumption could be removed in future deliverables. To preserve privacy assets against this threat, the CoNet architecture SHOULD not prevent the possibility of implementing privacy preserving mechanisms. The objectives of such mechanisms would be:

- (privacy objective 1.1) to verify the integrity and originality of named-data without disclosing the identity of the user that produced the data
- (privacy objective 1.2) to deliver named-data without disclosing the identity or the network location of the requesting user to intermediate un-trusted nodes

4.16 Scalability

Technical testing of the CONVERGENCE architecture will take place in special test-beds with restrictions on numbers of users, volumes of content and transaction rates. The scalability of the CONVERGENCE architecture can be defined as the possibility:

1) of deploying the system architecture in real life scenarios, under a predictable initial load
2) of further expanding the system as additional users and content are added in the future.

In this section we describe the conceptual framework we will use to assess the scalability of the CONVERGENCE architecture, first with reference to the CoMid and then to the CoNet.

4.16.1 CoMid scalability

CONVERGENCE content is described using Metadata included in VDIs. To be scalable, the CoMid has to allow an arbitrarily high number of users (all users of the current and the future Internet) to publish and retrieve an arbitrarily high number of VDIs. Given that publish-subscribe operations in the CoMid are implemented in a distributed way, by a set of CONVERGENCE peers, we can characterize the scalability of CONVERGENCE in terms of relations between the input load, and the capacity of the CONVERGENCE system, assuming the need to meet a set of performance targets. The input load can be characterized by: i) number of published VDIs; ii) number of active subscriptions; iii) overall rates of publication, subscription and search operations (e.g. the product of the number of users and the publication/subscription/search rate for each user). In addition we need to quantify the volume of "indexable" metadata contained in each VDI and the complexity of the queries used in search and subscribes operations. The capacity requirements for the CONVERGENCE system and for CONVERGENCE peers are given by: i) the number of peers; ii) the storage capacity of each peer; iii) the processing capacity of each peer; iv) inter-peer communication load (i.e. the messages that the peers need to exchange to support user requests). Finally, it is possible to set system performance targets in terms of: i) lookup time for search operations; ii) relevance of search results and notifications. In this conceptual model, CONVERGENCE may be considered as scalable if capacity requirements for the system are a linear or low order polynomial function of input load.

In practice, it may not be possible to achieve a full quantitative analysis of CONVERGENCE scalability. In what follows we will therefore adopt a qualitative approach.

The semantic overlay implements load-critical operations in the CoMid, including searching, indexing in pub/sub tables and circulation of metadata and is designed with scalability in mind. The design takes account of a number of critical conditions.

- **High network dynamism**. Peers can join and leave the network at any time. The search procedure has to perform normally, even if a peer fails.
- **High content dynamism**. Content stored in the network is highly dynamic due to frequent publications. and updates of previously published contents
- **Large search space.** Search and subscribe operations need the ability to find any publication on any peer in the CONVERGENCE network.

Given the need for scalability the design should have the following goals (see also [39]).

- The Semantic overlay must be **decentralized**: there should be no central entity controlling the search procedure. This avoids single-point-of-failure issues and ensures load-balancing between the peers that take part in the procedure.
- Semantic overlay must be **efficient**: peer resources and network bandwidth should be used with moderation.
- The Semantic overlay must be **scalable**: efficiency should be maintained when the volume of published content increases
- Semantic overlay must be **fault-resilient**: if a peer fails, search should be carried out normally.
- Semantic overlay must be **load-balancing**: the resources used for search should be distributed across all the peers of the network.

To minimize the search space, search is restricted to publications of a single semantic type, and signalling is propagated only to a subset of the overlay (the relevant fractal). However, CONVERGENCE also needs to address an additional requirement:

- Semantic overlay must be **flexible**: search queries provided by the user should be treated in a semantic manner, ensuring that results match the 'meaning' of the query.

In other words, CONVEGENCE has to deal with highly heterogeneous metadata based on different vocabularies and with ambiguous and inaccurate descriptions.

These requirements directly affect scalability: query expansion and the efficiency of SPARQL query engines efficiency needs to be evaluated in terms of their impact on the overall scalability of the CONVERGENCE system.

4.16.2 CoNet scalability

As far as concerns CONVERGENCE'S content-centric aspects, the CoNet has to scale up to the scale of current Internet, i.e. it should be capable of replacing the storage and retrieval capabilities of the current Internet.

To assess the scalability of CoNet we can use a similar conceptual framework to the one we used to analyze the CONVERGENCE publish-subscribe model. This means we need to define: i) the input load, ii) capacity requirements for the system, and iii) performance targets and characterize the relations between them. Here, the input load is defined by i) the number of different addressable items; ii) the number of *Principal Identifiers* (see sections 4.9 and 7.1.3) in the CoNet; and iii) the rate of user content requests. System capacity is defined in terms of i) storage and processing requirements for different types of CoNet nodes, ii) inter-node traffic to be supported. Performance targets will include i) content download rates (for non real-time content download); and ii) loss and delay (for real-time content transfer). As in the case of publish-subscribe, it is probably not feasible to analyze the scalability of the system in quantitative terms. We will therefore adopt a qualitative approach. Some details of our analysis will be included in the sections of this report dedicated to the design of the CoNet (e.g. Name-based routing: lookup-and-cache). Other details will be reported in future deliverables.

5 Overview of Application Level

The Application level provides the interface between users and CONVERGENCE and is the top level of CONVERGENCE from the user point of view. Applications will help users to perform tasks, such as searching, browsing and downloading VDIs. This requires that applications should be integrated with the CONVERGENCE middleware. In this section, therefore, we will provide an overview of middleware interactions with tools and application.

In general, CONVERGENCE applications access CONVERGENCE through one of three mechanisms.

- For middleware operations that are performed remotely, they use Elementary Services. Typical examples of such operations include identification, packaging and delivery of content.
- To execute a pre-specified chain of Elementary Services, they use aggregated services.
- To call a chain of engines that execute locally, they use a client Orchestrator to monitor the status of each engine and synchronize individual procedures.

CONVERGENCE will support two kinds of applications: Java standalone applications and web applications. While the former can make direct web service calls to access an elementary or an aggregated service, or to execute a local orchestration, the latter require an HTTP proxy that performs the necessary operations. In detail, when a browser-based (web) application makes request, this request is redirected to the proxy, which then decides whether it should make a CoNet call (e.g. in case of a requests for a middleware operation) or a plain TCP/IP based call (e.g. when it is necessary to fetch a web page). If an application requires streaming (e.g. the LMU podcast application or the FMSH video application) it relies on CoNet. In this case, the HTTP proxy will filter the client request and convert it to a call to CoNet. Finally, if the client side of the application (i.e. the browser) needs to make a local call to an orchestrator, it is passed through the HTTP proxy, as described in detail in deliverable D7.1. [40]

The Application level is split into two sub-levels: a Tools sub-level and a User Applications sub-level (see Figure 22). The Tools sub-level contains functional components that combine a subset of CoMid functionalities and can be reused by many applications but not all of them. Typical tools might include a VDI editor for creating new VDIs, a VDI search tool, a VDI browser tool and a VDI notification tool. In any given application, the Tools Layer is responsible for the interaction with CoMid; the Application level provides the interface to the user and visualizes search results in the format that best meets her requirements. This means that, for example, Photo VDIs or Video VDIs may appear differently in different applications.

By interacting with the CDS, tools can help the user to describe her resources. The CDS contains all the concepts available in the CONVERGENCE framework. Applications provide an interface to the service that users can exploit when they wish to describe a resource or define a set of search criteria.

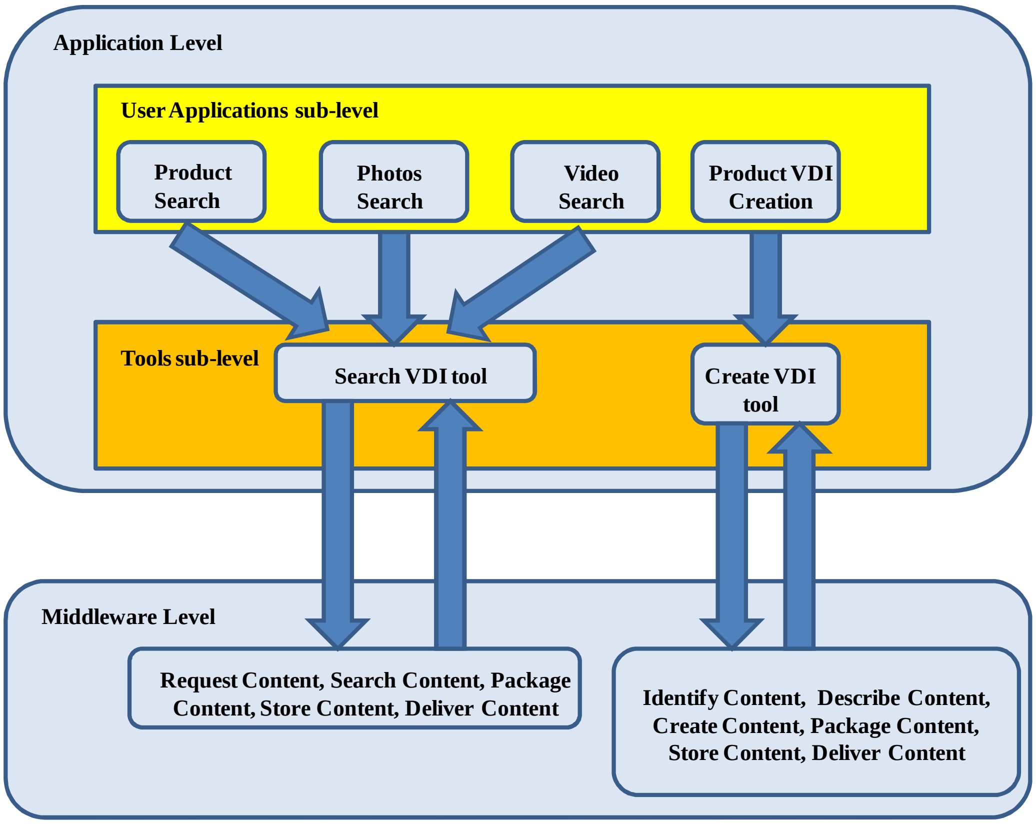

Application Level
User Applications sub-level
Product Search
Photos Search
Video Search
Product VDI Creation
Tools sub-level
Search VDI tool
Create VDI tool
Middleware Level
Request Content, Search Content, Package Content, Store Content, Deliver Content
Identify Content, Describe Content, Create Content, Package Content, Store Content, Deliver Content

Figure 22 - Interaction of Tools and Applications sub-levels with CoMid

6 Technical specification of CONVERGENCE Middleware (CoMid)

6.1 *CoMid overview*

The CoMid architecture is based on concepts developed in the MPEG-M (MPEG extensible Middleware) standard, an extension of the former MXM standard. CoMid functionality are implemented using Elementary Services in MPEG-M [see working drafts of ISO/IEC 23006-4 at http://mpeg.chiariglione.org/working_documents.htm#MPEG-M], and, when MPEG-M does not provide the necessary functionality, by new CONVERGENCE specific Elementary Services.

CoMid is based on a DDD (Describe, Discover, Distribute) paradigm. Its main purpose is to offer a set of standardized building blocks, APIs and protocols to describe, discover and distribute resources (VDIs) on the basis of “what” (metadata) they contain and offer. Higher-level Applications and Tools are built by assembling CoMid blocks into custom software, manipulated by users.

6.1.1 CoMid functionality

6.1.1.1 Describe functionality

Whenever a user produces a piece of multimedia content, or wants to market a good or service which is going to be governed and mediated through the CONVERGENCE system, the resource has to be described in a VDI. The VDI is a structured repository for all metadata and context information describing a specific resource. The CoMid “Describe” functionality is used to: i) access ontologies and directory services needed to tag resources; ii) create, identify and parse VDIs; iii) generate licensing information and contractual obligations which accompany the resource.

6.1.1.2 Discover functionality

Discovery of resources is crucial in large distributed networks. CoMid supports semantic searches over the VDI space (by semantic search, we mean a structured and multi-criteria search over the semantic part of VDIs. The result is a ranked list of resources, matching the search criteria). The CoMid Discover functionality uses semantic criteria to search for VDIs and to manage the topology of the VDI space, making it possible to perform queries efficiently.

6.1.1.3 Distribute functionalities

Users interact with the CONVERGENCE system through an asynchronous, publish/subscribe paradigm. The CoMid Distribute functionality implements pub/sub abstraction end-points towards the user, supports unpublish operations and implements content-based subscription to resources.

6.1.2 Elementary Services, Protocol Engines and Technology Engines

The CoMid is organized around the concept of Elementary Services (ESs) and implemented as Protocol Engines (PEs). Protocol Engines may call Technology Engines (TEs) to perform specific actions. PEs and TEs are assembled in a Service Oriented Architecture (SOA), based on the MPEG-M standard. Some of the Services needed by CoMid are directly available in the MPEG-M standard. Others will have to be specifically implemented within the CONVERGENCE framework. These services constitute a suite of building blocks facilitating the implementation of CONVERGENCE value chains (value chains are defined in MPEG-M as part of a business scenario). In such value chains, all devices are based on the same set of technologies, which they access via CoMid. These will include but not be limited to technologies standardized by MPEG. The use of a common set of technologies guarantees that all devices along the value chain can interoperate.

The section on Protocol describes the set of PEs implementing ESs in CoMid and the way they map onto functional areas of the CONVERGENCE middleware.

CoMid also introduces the concept of Entities, meaning the objects on which ESs act: VDIs (equivalent to Content in MPEG-M nomenclature), Devices, Events, Groups, Licences/Contracts, Services, and Users. By combining ESs and Entities, it is possible to describe a series of Operations. These are the basic building blocks in the CoMid SOA.

These Operations define a corresponding set of protocols and APIs that enable any user of the CONVERGENCE system to access those services in an interoperable fashion.

6.2 Orchestration and Aggregation

In CONVERGENCE, applications communicate with the middleware either by calling a protocol engine, which processes a request and calls technology engines to fulfil the request, or by calling a technology engine directly (as may occur when the technology engine is running on the local client device). Figure 23 shows a typical example where an application calls three protocol engines, PE1, PE2 and PE3 which in turn call remote technology engines and another technology engine which runs locally on the client device.

Using the middleware may require various remote calls and coordination of the results (see again Figure 23). This means that the work of a CONVERGENCE application developer may be rather complex. The same applies to the protocol engine provider who will need to coordinate technology engines.

CONVERGENCE will take advantage of the inherently, clear organization of protocol and technology engines, to support automatic execution of chains of engines. This will facilitate the work of the applications/middleware developer. To this end, we introduce the *aggregator* and the *orchestrator*, two middleware components that coordinate the execution of the atomic operations defined by the protocol and the technology engines.

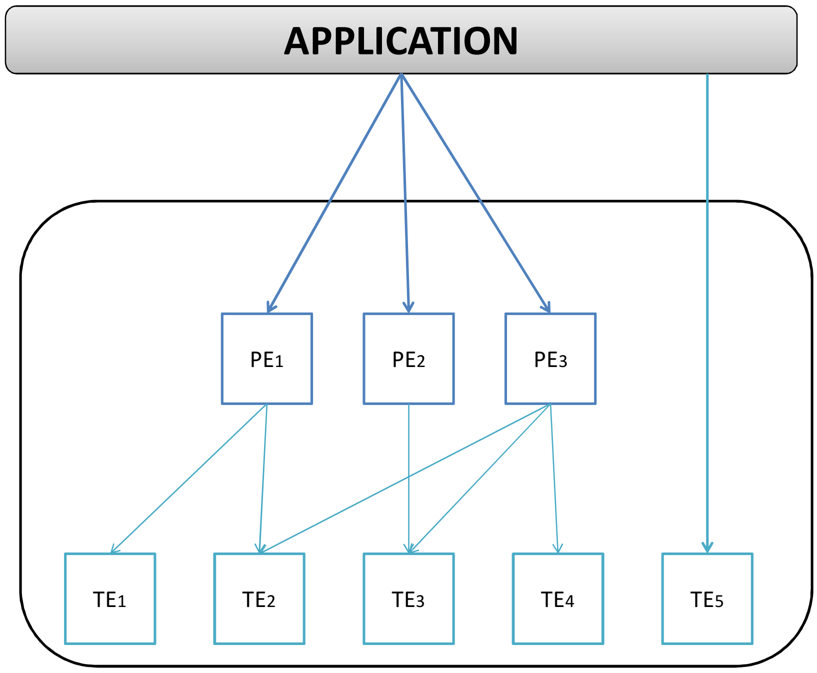


Figure 23 - Application Call to CoMid

Figure 24 illustrates how such an approach could be applied in action. Instead of calling each protocol engine separately, the application directly calls an *aggregation* of protocol engines, which exposes a new, (more) complex protocol, and performs the synchronization of the different engines.

Instead of calling and coordinating calls to various technology engines, the protocol engine relies on the *orchestrator*, which exposes a custom interface and relieves the protocol engine of the coordination of the calls.

The orchestrator and the aggregator coordinate and execute a chain of engines transparently. There are thus two ways of implementing them with the platform. The first is to develop each aggregation/orchestration separately, defining the interfaces through which they can be called and programming them in the same way as any other engine. The second is to implement the two components with workflow technology. More specifically we can represent chains of engines by BPMN (Business Process Model and Notation) workflows[4] . We can then use these workflows to dynamically create aggregations or orchestrations of engines. Given that workflows can incorporate executable processes, we can create BPMN processes which automatically make the necessary calls to the protocol and the technology engines, from within a general workflow environment.

[4] MPEG-M Part 5 [32] defines the workflow for the chain of protocol engines involved in Service Aggregation

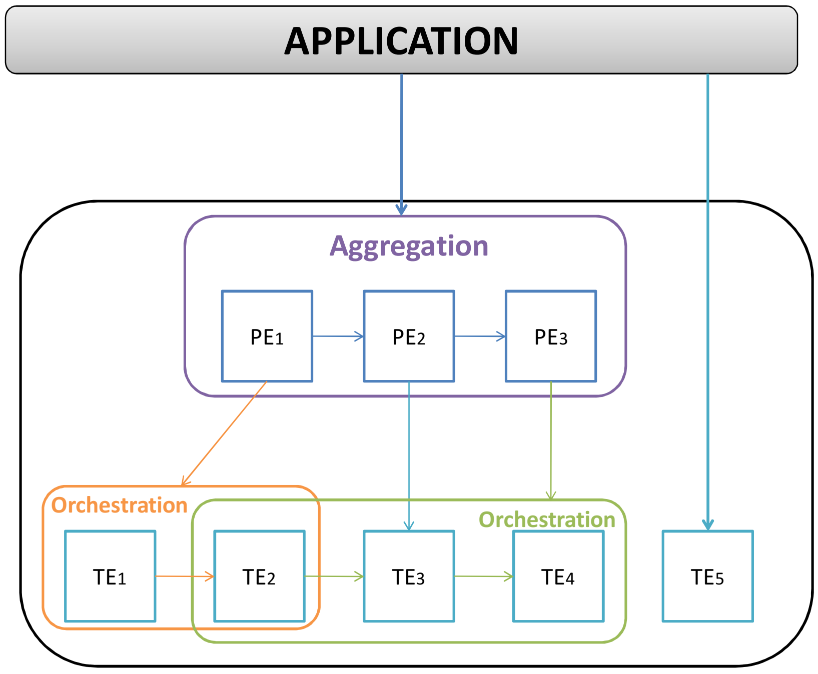


Figure 24 - Aggregation and Orchestration in CoMid

At the time of writing, we are considering both orchestration solutions. On the one hand, we are examining APIs for state of the art workflow engines (the second solution). On the other we are studying BPMN v2.0 as it is used to support service aggregation in MPEG-M Part 5 (the first solution) and have designed a general workflow environment that takes as input a BPMN v2.0 executable process (using a script task for each call to a method exposed by the technology engine), executes the process and returns the result to the caller.

6.3 Protocol engines

Protocol Engines used by CONVERGENCE (in *italics* engines extended by CONVERGENCE, in **bold** engines specified by CONVERGENCE)

Operation	*Entity*	*Definition*
Authenticate	Content	Allows a user to check the authenticity of a certain VDI on his/her device, by using the VDI itself and its digital signature. If the digital signature is not embedded inside the VDI itself, it has to be provided separately.
	Device	**Enables a user or any client application to authenticate the device where it is running. This way, the owner of the content to be used by the application can be sure that the device is trusted.**
	User	Allows the client to confirm a user's identity. This service can work with users belonging to different domains using a Single-Sign-On (SSO) mechanism.

Create	Content	Enables the creation of a Versatile Digital Item, by providing the data it contains, such as resources, descriptions, licenses etc.
	License	Enables the creation of a license, by providing the corresponding information expressed in a machine readable way. CONVERGENCE will support the MPEG-21 Rights Expression Language (REL) along possible extensions to be defined in D4.2.
Deliver	Content	Enables the client to request the delivery of a specified VDI according to terms and conditions, specified in a license. The Deliver Content service provider does not necessarily have to be the sender or the receiver of the content; it is just responsible for performing the transaction between the two parties
Describe	Content	Enables a user to generate, provide and retrieve descriptions of VDI content. The service provides 1. A protocol for setting the description of a VDI 2. A protocol for getting the description of a VDI **3. A protocol that facilitates the generation of semantic descriptions of VDI contents by providing entities from ontology models descriptions can be mapped to.** The Search Content Elementary Service uses these descriptions to perform search.
Identify	Content	Enables a User to obtain an identifier for a VDI or any of its components.
	User	**Enables a User to obtain an identifier. The identifier is stored in the service provider, so that it can later be used for retrieving it by a third party (e.g. an application) using the user's credentials**
Inject	**Content**	**Makes a VDI discoverable via semantic search operations. The operation performed by users or automatically by the system when a new VDI is published.**
Package	Content	Prepares a VDI for delivery, viz. creates a file out of the VDI, or a stream consisted of VDI fragments and binds it to the transport protocol
Present	**Content**	**Enables a user to set a style for the presentation of a VDI. The service provider will apply the style on the VDI and return the corresponding formatted document.**
Process	Content	Enables a user to modify the contents of a VDI. The result will be a new VDI, following the semantics of the update VDI (i.e. keeping the same sequence identifier as its predecessor).
	License	Enables a user to create a license, based on an already existing one. The new license may refer to different resources, different principals or be based on its predecessor template.
Request	Content	Enables a user to access a VDI.
Request	Event	Enables a user to ask for an Event Report Request associated with a given VDI, or the Event Reports associated with a given Event Report Request.
Revoke	**Content**	**Enables a user to unpublish/unsubscribe VDIs**
Search	Content	Enables a user to perform search on VDIs published in CONVERGENCE. The query may use the following classes of search criteria: • Unstructured criteria: free text description.

		• **Semi-structured criteria: property-value pairs where property is based on a model and value is free text.** • **Structured criteria: property-value pairs where property and value are both based on a model.** In the latter two cases, the user can use the Describe Content Elementary Service to request entities of existing models and construct the criteria for the query.
Store	Content	Allows for the transfer and the storage of a VDI in a local or remote device.
	Event	Enables a user to store an Event Report Request or an Event Report that may occur on a device.

See section 4.3 of deliverable D5.1 for complete details.

6.4 CDS TE

6.4.1 General description

The Community Dictionary Service Technology Engine (CDS TE) provides semantic services to the CONVERGENCE middleware. This section presents the main functional components of the CDS architecture. These components provide the following functionalities:

- Manage ontologies and dictionaries
- Perform queries against CDS's knowledge base for
    - Exploring ontology entities
    - Expanding semantic descriptions
    - Expanding semantic queries

6.4.2 Main components and their interaction

The following Figure 25 depicts the architecture of the CDS TE.

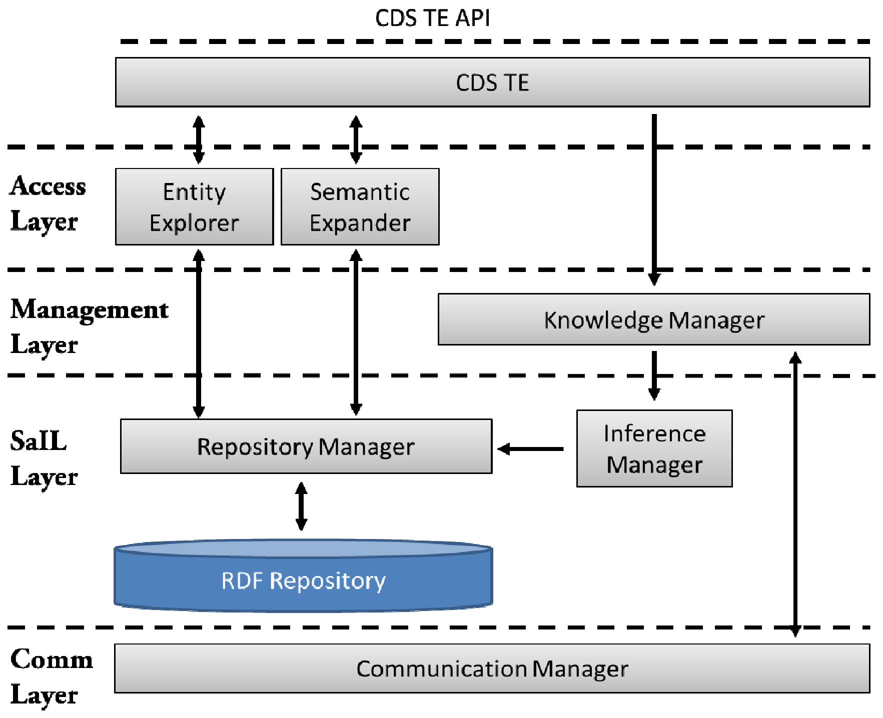


Figure 25 - Architecture of the CDS TE

The following table details the function of the layers depicted in the figure.

Components	**Function**
Access Layer	The access layer involves interfaces that exploit knowledge maintained by the CDS • Semantic Expander: this interface defines methods for building materialized semantic descriptions, for building semantically equivalent semantic descriptions and for building semantically equivalent queries • Entity Explorer: this interface defines methods for querying the repository for ontology entities and building a result set
Management Layer	The management layer involves interfaces that manage the ontologies and the dictionaries that CDS maintains. • Knowledge Manager: this interface defines methods for loading, unloading and retrieving ontologies and dictionaries from CDS knowledge base.
Storage and Inference Layer	The storage and inference layer involves interfaces for managing the knowledge

(SaIL)	repository and reasoning over it. • Repository Manager: this interface abstracts the storage format used for the knowledge base and defines methods for querying it, as well as adding and removing statements. • Inference Engine: this interface provides reasoning support over the knowledge base. It defines methods for adding/materializing and removing statements from the knowledge base and for checking the consistency of it.
Communication (Comm) Layer	The communication layer involves interfaces for the communication between CDS servers. • Communication Manager: this interface defines methods for fetching ontologies and dictionaries from other CDS servers. It also provides an entry point for other CDS servers to access it.

6.5 CoNet TE

The CoNet Technology engine provides the CONVERGENCE Middleware with access to the features of the CoNet, based on Content Centric Networking. As described in section 3, the CoNet is a part of the Computing Platform level. The internals of the CoNet are described in section 7.1. In this section we describe the API offered by the CoNet technology engine to the other elements of the CONVERGENCE Middleware.

6.5.1 Main components and their interaction

The CoNet technology engine is a Java package that implements the functionality described by the CoNet API. The Java package also includes the classes that represent the CoNet identifiers (NID, LID, see section Network).

6.6 Digital Item TE

The *DI Engine* interface defines methods for operating on ISO/IEC 21000-2 Digital Item Declaration (DID) data structures. From classes implementing the *DI Engine* interface it is possible to obtain instances of classes performing the main functionalities of this MXM Engine:

- Classes to create Digital Items
- Classes to edit a Digital Item
- Classes to access data contained in a Digital Item

6.6.1 Main Components and their Interaction

The following table details the main components of the Digital Item Engine and their functions.

Components	Function
DI Creation	Digital Item creation involves the following interfaces: • DICreator: an interface defining methods to create a Digital Item Declaration and to add a digital signature to it • ItemCreator: an interface defining methods to create a didl:Item after setting all its main properties • DescriptorCreator: an interface defining methods to create a didl:Descriptor after setting all its main properties • ComponentCreator: an interface defining methods to create a didl:Component after setting all its main properties • ResourceCreator: an interface for creating a didl:Resource after setting all its main properties
DI Editing	Digital Item editing involves the following interfaces: • DIEditor: an interface defining methods to edit a Digital Item • ItemEditor: an interface defining methods to edit a didl:Item.
DI Access	Digital Item parsing involves the following interfaces: • DIParser: an interface defining methods to parse a Digital Item and retrieve the information contained unit.

6.7 Event Report TE

The Event Report Technology Engine (TE) defines methods for operating over ISO/IEC 21000-15 Event Reporting data structures. It will implement methods to perform the following operations:

- Create Event Report Requests (ER-R)
- Add / modify data to / from an ER-R
- Parse and Retrieve data from an ER-R

- Create Event Reports (ER)
- Notify users with ERs
- Parse and Retrieve data from an ER

6.7.1 Event Report Request (ER-R) message

The Event Report Request message is composed of three main sections as shown in the table below. Each main section consists of several parts which are described next.

Main Section	Content
ER-R Descriptor	*ER-R Identifiers* – uniquely identifying an ERR *ER-R Lifetime* – provides information about the lifetime of the ERR *ER-R History* – provides an audit trail of the ERR. *ER-R Priority* – provides information on the priority of an ER.
ER Descriptor	*ER Identifier* – providing an identifier to be used in the ER(s) created based on this ER-R. *ER Access control information* – provides information about which Peers and/or Users are allowed to access the ER and about the mechanism these access rules are enforced. *ER Fields* – provides information about the fields to be contained in the ER. *ER Formats* – The syntax and semantics of these fields. *ER Delivery Attributes* – provides information such as the intended recipient(s) of the ER and the delivery mechanism.
Event Conditions Descriptor	Time-based Conditions Condition(s) on User operations on a VDI Condition(s) on other operations or environment

Table 7 – Event Report Request message structure

6.7.2 Event Report (ER) message

The basic model of Event Reporting indicates that Events that need to be reported are represented as an Event Report specified by an Event Report Request. An Event Report message is composed of four main sections as shown in the table below. The Event Report is automatically generated by the Event Report TE as specified in the associated Event Report Request.

Main Section	Content
ER	*ER Identifier(s)* – uniquely identifies the ER

Descriptor	*ER Format* – provides information about the format in which the ER Data is delivered. *ER Access control information* – provides information about which Peers and/or Users are allowed to access the ER and about the mechanism these access rules are enforced. *ER Status* – Provides information on whether the Peer was able to properly generate the ER *ER History* – provides an audit trail of the ER *ER Priority* – provides information on the priority of an ER
ER Source	The information about the original source of ER-R that triggers the Report.
ER Data	Report Data which is needed in the associated ERR.
Embedded ER-R	The other ER-R which related with this ER.

Table 8 -Event Report message structure

6.7.3 Main components and their interaction

The following Figure 26 depicts the architecture of the Event Report TE based on MPEG-21 Event Reporting.

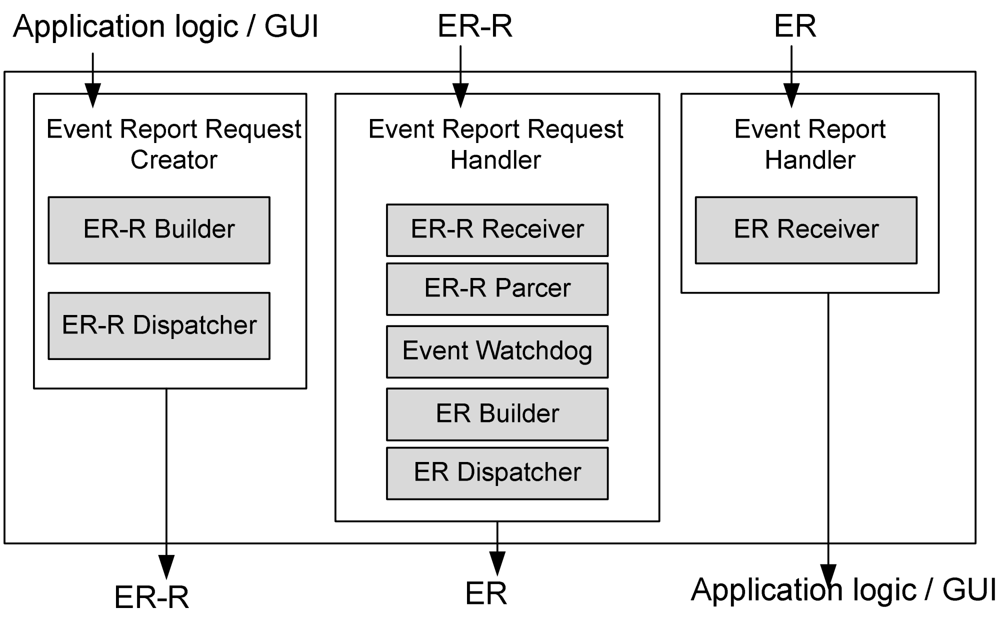


Figure 26 - Architecture of Event Report TE

An Event Report Request is created by the **Event Report Request Creator** component which cooperates with the application logic and/or User action using user interface (GUI).

Upon receiving an ER-R message, the **Event Report Request Handler** component parses the Event Reporting message and monitors an Event occurrence. Upon fulfilment of a set of events, Event Reports are generated by the *ER Builder* subcomponent and are sent to the recipients specified in associated Event Report Requests.

After receiving an ER message, the ***Event Report Handler*** component parses that message, then displays and stores information contained in the message.

Figure 27 shows the detailed structure of the Event Report Request Handler component and describes the interaction among its five elements on receiving an Event Report Request.

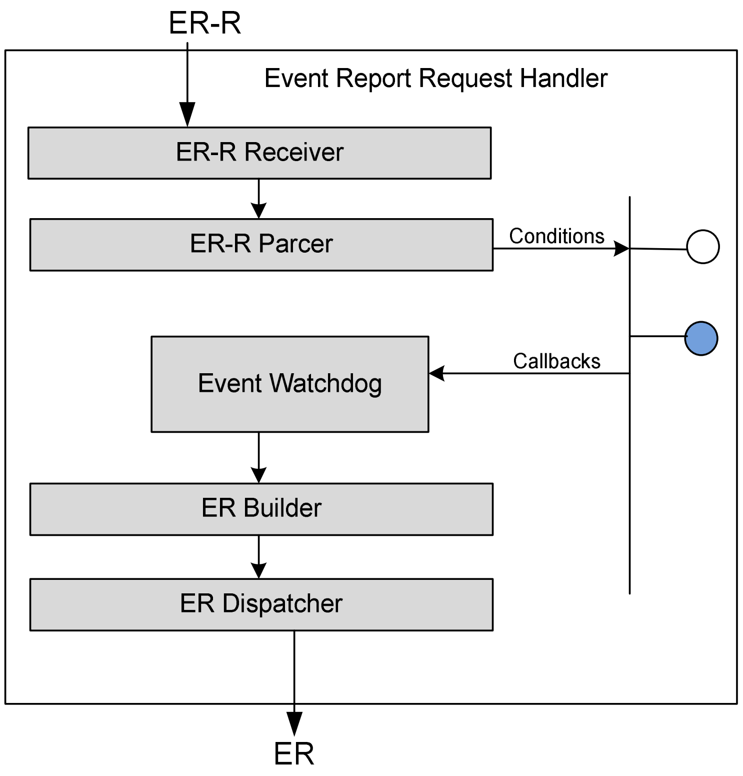


Figure 27 - Detailed structure of the Event Report Request Handler

The elements are:

ER-R Receiver – receives an ER-R from another Peer;

ER-R Parser – interprets ER-Rs;
Event Watchdog – monitors Events and detects when ER-R conditions have been fulfilled;
ER Builder – assembles reportable Event data and creates an Event Report;
Event Report Dispatcher – takes an Event Report and sending it to designated recipient Peers.

6.8 *Media Framework TE*

The Media Framework TE is a high level engine, grouping several media specific engines (e.g. Video, Image, Audio and Graphics Engines). Each exposes APIs to create (encode) and access (decode) their respective elementary streams. The Media Framework TE also implements common functionalities (independent of media type) such as resource loading and saving. The Media Framework TE has two interfaces:

- An interface for accessing content (Access)
- An interface for creating content (called Creation)

A typical implementation of the Access interface of the Media Framework TE loads a resource, demultiplexes it, checks the type of the elementary streams within the resource and calls the associated elementary stream access engines.

A typical implementation of the Creation interface of the Media Framework TE calls the associated elementary stream creation engines, initializes them with encoding parameters and saves the multiplexed resource.

6.8.1 Main components and their interaction

The main components of the Engine are reported in the following table.

Components	**Function**
Creation	Creates and handles a given resource.
Access	Consumes a given resource (e.g. by rendering it on a visual panel

6.9 *Match TE*

6.9.1 General description

The Match Technology Engine (TE) is responsible for performing matches between subscriptions and publications. To support the CONVERGENCE content-based publish/subscribe paradigm, matches are based on the descriptions of content, as detailed in previous sections. The Match TE identifies Semantic descriptors within the VDI structure and fires the corresponding parser.

The Match TE maintains separate repositories to handle the descriptors that are to be matched, depending on whether they come from subscriptions or publications.

6.9.2 Main components and their interaction

The **Match TE** is used by the CoMid to match subscriptions and publications whenever it receives a publication or a subscription (see Figure 28).

On receiving a subscription, the **Subscription Manager** is called. The Subscription Manager adds the subscription to the subscription tables it maintains. These tables index subscriptions according to their semantic type, i.e. according to the fractals they participate in (for more info see section on Semantic). To store the subscription, it calls the **Repository Manager**. Finally, it calls the **Matching Engine** to check the subscription for matching publications. The Subscription Manager also controls the deletion of subscriptions from subscription tables and the relevant database when they expire.

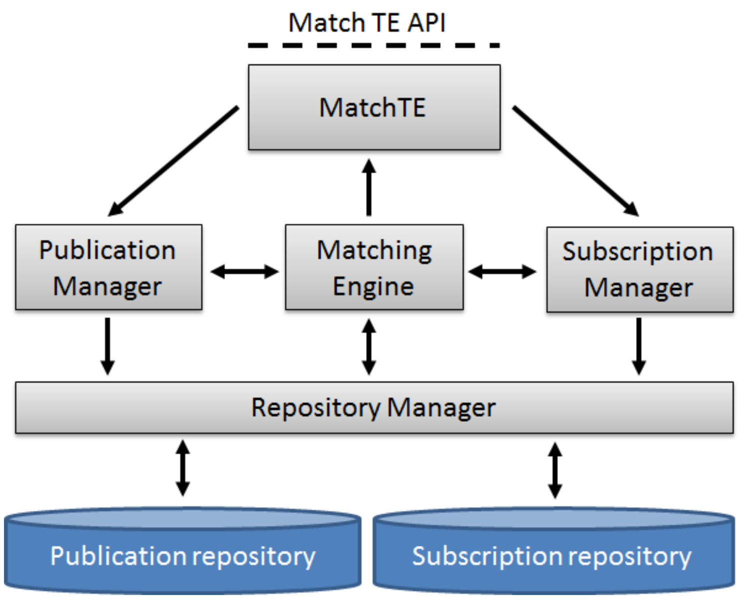


Figure 28 - Architecture of the Match TE

On receiving a publication, the **Publication Manager** is called. The Publication Manager adds the publication to the publication tables it maintains, which function in the same way as the tables for the Subscription Manager. Finally, it calls the **Matching engine** to check the publication for matching subscriptions. The Publication Manager also controls the deletion of publications when they expire, and the revocation of publications from publication tables and the relevant database.

As mentioned earlier, subscriptions and publications are stored in database. For this task the **Repository Manager** is called. The Repository Manager maintains databases for the storage of publications and subscriptions, and is also responsible for querying the relevant databases whenever necessary.

Finally, the **Matching Engine** performs the match operation and returns the results. On receiving a request to check for matches of a subscription, the Matching Engine retrieves the relevant database from the **Publication Manager**, checks the query and forwards it to the Repository Manager. It then returns the resulting matches to the system.

On receiving a request to check for matches of a publication against existing subscriptions, the Matching Engine retrieves the relevant database of subscriptions from the **Subscription Manager** and checks each subscription against the newly received publication.

6.10 Metadata TE

The Metadata TE interface defines methods for operating over metadata structures. Classes implementing the Metadata TE interface act as factories creating instances of classes performing the following functionalities:

- Create metadata structures, by means of the MetadataCreationEngine
- Access data contained in metadata structures, by means of MetadataAccessEngine

6.10.1 Main Components and their Interaction

The main components of the Engine are reported in the following table.

Components	**Function**
Metadata creation	Creating metadata structures involves the following interfaces: • MetadataCreator: a super interface defining a basic method to create a metadata structure; in the future CONVERGENCE will extend this interface • GenericMetadataCreator: an interface defining methods to create generic metadata structures in any format, depending on the specific implementation of the Metadata Engine of choice. • Mpeg7Creator: an interface defining methods to create MPEG-7 metadata structures. • AbstractCreator: an interface defining methods to create an mpeg7:Abstract element • CreationCoordinatesCreator: an interface defining methods to create an mpeg7:CreationCoordinates element • CreationDescriptionCreator: an interface defining methods to create an mpeg7:CreationDescription element • CreatorCreator: an interface defining methods to create an mpeg7:Creator

	element • GenreCreator: an interface defining methods to create an mpeg7:Genre element • ParentalGuidanceCreator: an interface defining methods to create an mpeg7:Creator element • TitleMediaCreator: an interface defining methods to create an mpeg7:Creator element
MetadataAccess	Accessing metadata structures involves the following interfaces: • MetadataParser: a super interface to be further extended, defining a basic method to create a metadata structure • GenericMetadataParser: an interface defining methods to parse a generic metadata structure in possibly any format depending on the specific implementation of the MetadataEngine of choice. • Mpeg7Parser: an interface defining methods to parse MPEG-7 metadata structures. • AbstractParser: an interface defining methods to parse an mpeg7:Abstract element • CreationCoordinatesParser: an interface defining methods to parse an mpeg7:CreationCoordinates element • CreationDescriptionParser: an interface defining methods to parse an mpeg7:CreationDescription element • CreatorParser: an interface defining methods to parse an mpeg7:Creator element • GenreParser: an interface defining methods to parse an mpeg7:Genre element • ParentalGuidanceParser: an interface defining methods to parse an mpeg7:ParentalGuidance element • TitleMediaCreator: an interface defining methods to parse an mpeg7:TitleMedia element

6.11 MPEG-21 File TE

The MPEG21 File TE interface defines methods for operating over MPEG-21 File Format files. The classes implementing the MPEG21File TE interface make it possible to obtain:

- instances of classes to create an MPEG-21 file – these class instances implement the MPEG21 FileCreator interface.

- instances of classes to access data from an MPEG-21 file – these class instances implement the MPEG21 FileAccess interface.

6.11.1 Main Components

The main components of the Engine are reported in the following table.

Components	Function
MPEG21 File Creation	Creating an MPEG-21 file involves the following interfaces: • MPEG21FileCreator: an interface defining methods to create an MPEG-21 file
MP21 File Access	Accessing an MPEG-21 file involves the following interfaces: • MPEG21FileAccess: an interface defining methods to access an MPEG-21 file

6.11.2 MPEG-21 File Format

MPEG-21 files are structured in accordance with MPEG-21 Part 9 (File Format). They use the structural definition of box-structured files, as defined in the ISO Base Media File Format, but not the definitions for time-based media.

An MPEG-21 file stores an MPEG-21 digital item, as well as some or all of its ancillary data (such as images, movies, or other non-XML data). In it, a file-level meta-box is used to hold an MPEG-21 Digital Item Declaration (DID) and a list of attached resources (which may have local names, and may be located within the same file or in another file).

6.12 Overlay TE

The Overlay TE is responsible for distributing content to a set of peers which are interested in that type of content. It is used by the publish-subscribe mechanism of CONVERGENCE to diffuse publication and subscription VDIs to a group of peers that have already published or subscribed to content of that type. It has two fundamental components: the topology management component and the content propagation component. The topology management component is used by each peer so that it can keep a consistent view of the system that it needs to propagate to. The content propagation component is used by the peers to diffuse content to a set of target peers in the system.

6.12.1 Main components and their interaction

The main components of the Engine are reported in the following table.

Components	Function
Topology Management	This component is related to the registry, as this contains the topology view of each peer. There two interfaces defined to serve the purpose of this component:

	• RegistryCreator: this interface defines methods needed to register to a fractal, notify other fractal members of the registration and advertise the registry to the network, in order to make it retrievable by new, incoming peers. • RegistryParser: this interface defines methods necessary for parsing the registry and extracting the necessary information for maintaining topology and propagating messages to the fractal.
Content Propagation	This component defines interfaces for handling the propagation of publication and subscription messages, as well as the buffers for storing these messages into the peers: • PropagationMessageHandler: this interface defines methods for sending, receiving and consuming a message (e.g. a publication VDI) over the overlay. • MessageCreator: this interface defines methods for creating an overlay message. • MessageParser: this interface defines methods for parsing the different parts of an overlay message. • MessageKeeper: this interface defines methods needed to store a message in the local buffers so that it will be available for (i) the next propagation round and (ii) perform the matching.

6.12.1.1 Message Format

An Overlay TE message consists of the header and the payload, as depicted in Figure 29. The header has four fields:

- The path with the peers that the message has visited so far (before sending the message, any peer which is not the initiator adds itself to the list).
- The fractals (in case more than one fractals are defined in this field, each peer will consider the intersection of them) where the message is propagated.
- The time to live of this message, i.e. how many propagation rounds will have to take place.
- An operation code indicating the type of the message (publication, subscription or discovery gossiping).

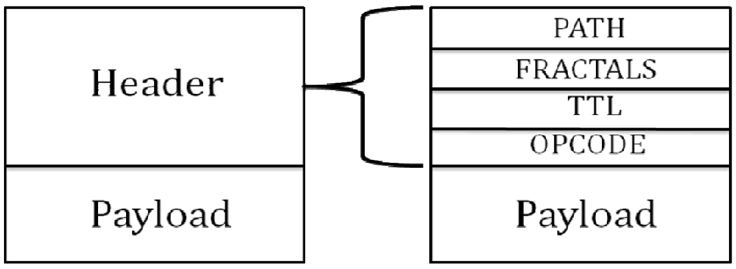


Figure 29 - Overlay TE Message Format

Below we present the XML Schema for the overlay propagation message.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
       xmlns:didl="urn:mpeg:mpeg21:2002:02-DIDL-NS"
       targetNamespace="http://www.ict-CONVERGENCE.eu/schemas/gorestMessage"
       xmlns:gorest="http://www.ict-CONVERGENCE.eu/schemas/gorestMessage"
       elementFormDefault="qualified">
       <xs:import namespace="urn:mpeg:mpeg21:2002:02-DIDL-NS"
schemaLocation="http://standards.iso.org/ittf/PubliclyAvailableStandards/MPEG-
21_schema_files/did/didl.xsd"/>
       <xs:element name="gorestMessage" type="gorest:gorestMessageType"/>
       <xs:complexType name="gorestMessageType">
              <xs:sequence>
                     <xs:element name="header" type="gorest:headerType"/>
                     <xs:element name="payload" type="gorest:payloadType"/>
              </xs:sequence>
       </xs:complexType>
       <xs:complexType name="headerType">
              <xs:sequence>
                     <xs:element name="TTL" type="xs:int"/>
                     <xs:element name="opcode" type="xs:int"/>
                     <xs:element name="path" type="gorest:pathType"/>
                     <xs:element name="fractals" type="gorest:fractalsType"/>
              </xs:sequence>
       </xs:complexType>
       <xs:complexType name="payloadType">
              <xs:choice>
                     <xs:element name="publication_vdi" type="didl:DIDLType"/>
                     <xs:element name="subscription_vdi" type="didl:DIDLType"/>
                     <xs:element name="discovery_message"
type="gorest:registrationMessageType"/>
              </xs:choice>
       </xs:complexType>
       <xs:simpleType name="pathType">
              <xs:list itemType="xs:string"/>
       </xs:simpleType>
       <xs:simpleType name="fractalsType">
              <xs:list itemType="xs:string"/>
       </xs:simpleType>
       <xs:complexType name="registrationMessageType">
              <xs:sequence>
                     <xs:element name="peerId" type="xs:string"/>
                     <xs:element name="overlaySAP" type="xs:anyURI"/>
                     <xs:element name="leaveDate" type="xs:dateTime"/>
```

```
            </xs:sequence>
      </xs:complexType>
</xs:schema>
```

The XML schema of the peer registry is as follows.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<xs:schema xmlns:gore="http://www.ict-CONVERGENCE.eu/schemas/registry"
      xmlns:xs="http://www.w3.org/2001/XMLSchema"
      elementFormDefault="qualified"
      targetNamespace="http://www.ict-CONVERGENCE.eu/schemas/registry">

<xs:element name="registry" type="gore:registryType"/>
<xs:complexType name="registryType">
<xs:sequence>
      <xs:element name="ontologyLocation" type="xs:anyURI"/>
      <xs:element name="peerIds" type="gore:peerList"/>
      <xs:element name="overlaySAPs" type="gore:overlaySAPList"/>
      <xs:element name="leaveDates" type="gore:leaveDateList"/>
</xs:sequence>
<xs:attribute name="fractal" type="xs:string"/>
<xs:attribute name="fractalMembers" type="xs:int"/>
</xs:complexType>

<xs:simpleType name="peerList">
      <xs:list itemType="xs:string"/>
</xs:simpleType>

<xs:simpleType name="overlaySAPList">
      <xs:list itemType="xs:anyURI"/>
</xs:simpleType>

<xs:simpleType name="leaveDateList">
      <xs:list itemType="xs:dateTime"/>
</xs:simpleType>
</xs:schema>
```

6.13 REL TE

A Rights Expression Language (REL) is a machine-readable language that declares rights and permissions. The MPEG REL, as defined by ISO/IEC 21000-5, provides flexible, interoperable mechanisms to support transparent and augmented use of digital resources throughout the value chain in a way that protects the digital resource and honours the rights, conditions, and fees specified for it. The REL Engine interface defines methods for operating over Rights Expression Language (REL) data structures. Classes implementing the REL Engine interface act as factories creating instances of classes performing the following functionalities:

- Create Rights Expressions
- Access data contained in Rights Expression
- Authorise users to exercise rights

- Copy and move digital resources according to pre-determined rules

6.13.1 Main components and their interaction

The main components of the Engine are reported in the following table.

Components	Function
Rights Expression creation	Creating a REL statement involves the following interfaces: • LicenseCreator: an interface defining methods to create an r:license element. The output is an xml file containing the created license • GrantCreator: an interface defining methods to create an r:grant • DigitalResourceCreator: an interface defining methods to create an r:digitalResource (e.g. encryption/ decryption/ format) • ProtectedResourceCreator: an interface defining methods to create an m1x:protectedResource • IdentityHolderCreator: an interface defining methods to create an m1x:identityHolder • IssuerCreator: an interface defining methods to create an r:issuer • KeyHolderCreator: an interface defining methods to create an r:keyHolder
Rights ExpressionAccess	Parsing a REL statement involves the following interfaces: • LicenseParser: an interface defining methods to parse an r:license element • GrantParser: an interface defining methods to parse an r:grant • DigitalResourceParser: an interface defining methods to parse an r:digitalResource • ProtectedResourceParser: an interface defining methods to parse an m1x:protectedResource • IdentityHolderParser: an interface defining methods to parse an m1x:identityHolder • IssuerParser: an interface defining methods to parse an r:issuer • KeyHolderParser: an interface defining methods to parse an r:keyHolder
Authorisation	Authorizing a user to exercise a right involves the following interfaces: • AuthorisationManager: an interface defining methods to authorise a user to exercise a right and retrieve information from the validation operation • AuthorisationResult: an enumeration defining possible result of an authorisation.

6.14 Security TE

The SecurityEngine interface defines security-related methods. Classes implementing the SecurityEngine interface implement methods providing the following functionalities:

- Create new credentials and manage certificates
- Generate symmetric keys and encrypt/decrypt data
- Store confidential information such as licenses and keys in the secure repository.
- Certify the integrity of MXM tools

6.14.1 Main components and their interaction

The main components of the Engine are reported in the following table.

Components	Function
CertificateManager	The certificate manager involves the following interfaces: CertificateManager, to generate private/public keys, import and export of public keys and certificates in various formats, perform cryptographic services, secure storage and retrieval of information, generation of keys, signature calculation and validation, etc.
KeyManager	The key manager involves the following interfaces: • KeyManager, to generate symmetric keys, providing generation of keys, hashes, signature calculation and validation, etc.
SecureDeviceManager	The Secure Device Manager involves the following interfaces: • SecureDeviceManager, to certify and verify the integrity of MXM Devices, requesting the calculation of a fingerprint of the device with the hardware software installed, and the verification of these values.
SecureRepositoryManager	The Secure Repository Manager involves the following interfaces: • SecureDeviceManager, to store, retrieve and manage confidential information in the secure repository.

7 Technical specification of Computing Platform level

7.1 Network component

Network level functionalities are provided by the CONVERGENCE Network (CoNet). CoNet is a content-centric inter-network that provides users with a network access to remote named-resources, rather than to remote hosts. Named-resources can be either data[5] (in the following referred to as “named-data”) or service-access-points[6] (“named-service-access-points”), identified by a network-identifier (a name).

The name of a resource is its *network-identifier* (NID).Unlike Internet URLs, which include information about “where” a resource is located, CONVERGENCE NIDs do not necessarily contain a reference to a resource’s location. A network-identifier is an *anycast* address: a system may contain multiple replicas of the same named-resource. Network functionality will connect a user to the “best” replica (e.g., the closest one). For example:

- A PDF copy of the TIMES newspaper for Sept-30-2010 could be a resource identified by “the times: Sept-30-2010”;
- The service-access-point for an SQL database service provided by the Foo company could be a resource identified by the name “Foo: sql-database”;
- The service-access-point for an MXM Content-provider device owned by the Foo company could be a resource identified by the name “Foo: mxm-content-provider”.

As it handles named-resources through their network identifiers, rather than IP addresses, the network infrastructure is aware of the resources it is handling. This awareness can be exploited to support anycast routing, resource replication, and in-network caching [4] (if resources are data and not points of access to a service). These mechanisms are very useful but not supported by current IP networks[7]. The possibility of accessing resources without having to specify their location, simplifies mobility issues. It also means that resource names can be portable and do not depend on the service provider providing the resource.

The CoNet allows users (e.g., end-users and providers) to advertise and revoke their named-resources. A resource can be replicated in different geographical locations by adopting the same network-identifier.

[5] Named-Data include: documents, video, images, structured information, VDIs. The Network level is a general purpose one, as it can handle any kind of data and not only data generated by the CONVERGENCE system.

[6] A named-service-access-point is a network endpoint through which an upper layer entity (e.g., a server or a client) sends and receives data. In the actual Internet, for instance, TCP port n. 80 is the default service-access-point for HTTP servers.

[7] Some partial support is provided by proprietary systems, such as Content Delivery Networks (e.g. Akamai).

Replication enables users to exploit resources that are closer to their location and improves reliability of access. The CoNet provides *secure distribution* of resources by ensuring that the network-identifier cannot be forged and that receivers can verify the validity and provenance of named-resources [2][5]. Network functionalities can autonomously cache transiting resources [4], if this is allowed by security rules and by the nature of the resource[8].

When a named resource is named-data, the CoNet provides the means to deliver it to intended recipients. It can also facilitate the delivery of the named-data through replication/caching.

When a named resource is a named-service-access-point, the CoNet provides the means to exchange data between a requesting upper layer entity and the upper layer entity addressed by the named-service-access-point. This responds to a different need with respect to the distribution of named-data mentioned above. In the case discussed here, upper layer entities want to exchange data that do not need to be named and do not need to be made accessible and identified in the network. This functionality is needed by "traditional" client/server services (e.g. HTTP, POP, SMTP) and middleware services (e.g. transport of middleware service data), which do not assign names to their data. To support this need we introduce the concept of "*un-named-data*", i.e. data not identified by a NID. The named-sap case can be extended to multicast; in this case the NID of a named-sap has a multicast meaning, rather than an anycast one.

The CoNet thus offers two mechanisms for the delivery of data.

1. When users need to access named-resources that are data (e.g., documents, VDIs, files, etc.), the CoNet allows them to download actual content. In this case, the user asks the network to provide a resource with a given network-identifier and the network provides her with the actual content, without involving any other upper layer functionality. This modus operandi follows the *content-centric networking* paradigm recently proposed in the literature.
2. When it is necessary to support service sessions for upper layer entities (e.g. a client-server couple), the CoNet couples the local upper layer entity with the named-service-access-point for the "best" remote upper layer entity providing the service (via anycast routing) and goes on to support an interactive exchange of un-named-data between these two upper layer entities. In the case of a client-server service session, for example, the un-named-data are upper layer data (e.g. HTTP, SMTP, POP, SQL, MXM messages) exchanged between a local client and a remote server (e.g. an HTTP server, a middleware device etc.). Thanks to this functionality, the CoNet can natively support most current Internet services, existing middleware protocols and any service that requires point-to-point bidirectional interaction or point-to-multipoint communications. This *extension* to the capabilities of a "plain" content-centric network [3] allows the CoNet to support not only content retrieval but also more traditional services.

[8]It is possible to cache data but not points of access to a service.

Access to a named-resource involves two steps: i) anycast routing of a user "request" to the best CoNet node providing the named-resource and ii) unicast routing of the "response" from the selected CoNet node toward the requesting CoNet user. Anycast routing uses the network-identifier of the named-resource, as previously discussed. In unicast routing, endpoints are identified by a *location-identifier* (LID) – the equivalent of the IP address and port number on the current Internet. The use of location-identifiers makes it possible to support sessions that require interactive exchange of data between a "client" and a "server". After the first phase of anycast routing, when the CoNet selects the best server, further communications between the client and the server can be routed using their location-identifiers; in fact, if the client application tried to reuse the (anycast) network-identifier, the CoNet might select a different server, interrupting the session[9.10].

7.1.1 CoNet Architecture

The CoNet is an (inter-)network layer that provides users with a network access to remote named-resources. The main features of CoNet can be summarized as follows.

- It is stateless: network nodes do not maintain information on the ongoing communications.
- It limits the size of name-based routing tables by caching only a subset of all possible routes; missing routing entries are looked up in a name-system and then cached.
- It can be integrated in existing IP networks by using a new header option, which makes IP content-aware [6]. In this case, the nodes could use hybrid routing tables containing both IP network addresses and names. However, CoNet also supports the clean-slate or overlay deployment approaches.

CoNet is a system that interconnects CoNet Sub Systems (CSSs) (see Figure 30[11]). A CSS contains CoNet nodes and exploits an under-CoNet technology to transfer data among CoNet nodes. A CSS could be:

- A couple of nodes interconnected by a point-to-point link, e.g. a PPP link or a UDP/IP overlay link.

[9]One way to implement the location-identifier (without exploiting IP) would be to adopt a *link-layer source-routing* approach. In this case, the structure of the location-identifier would represent the sequence of link-layer interfaces to be followed from the CoNet node (Server) to the requesting CoNet user, i.e. on the reverse path. This sequence is built during the preceding anycast routing phase. The location-identifier has a *temporary* meaning; it is assigned in a distributed way, and has no impact on the routing-plane of the CoNet, which is concerned only with network-identifiers. *Link-layer source-routing* is the approach adopted by the PSIRP FP7 project (http://www.psirp.org/).

[10] It has not yet been decided whether the CONVERGENCE system will support also service sessions between two NIDs, when it happens that NIDs are unicast addresses and thus univocally identifies a service access point.

[11] Please note that Figure 30 is identical to Figure 14 and is replicated here for the reader's convenience.

- A layer-2 network, e.g. Ethernet, or a layer-3 network, e.g. a private/public IPv4 or IPv6 network, or a whole IP Autonomous System, or even the whole current Internet.

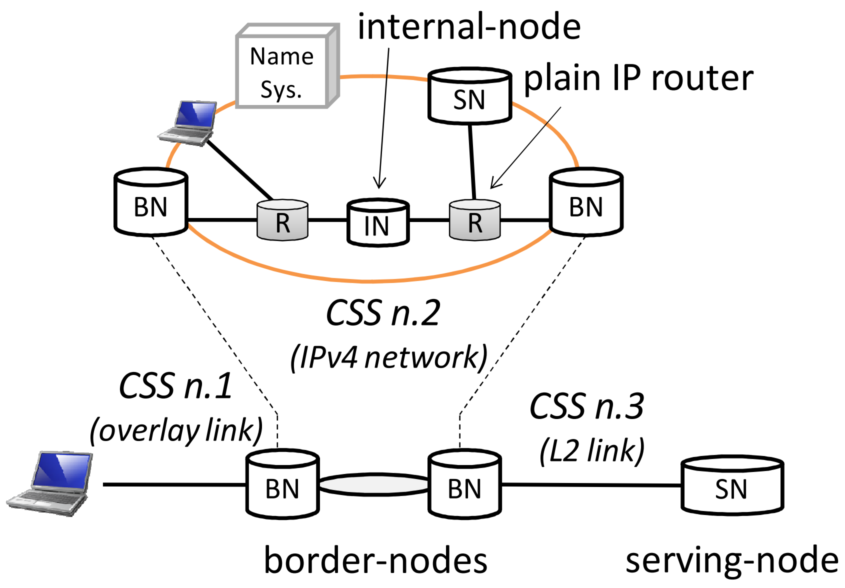


Figure 30: CoNet Architecture

The devices within a CSS use an autonomous and homogeneous under-CoNet addressing space and, if necessary, an interior under-CoNet routing protocol (e.g. [13]).

CSSs can be defined rather freely. For instance, if CoNet protocols are implemented only in user equipment, interconnected by the current Internet, we have only one CSS: the current Internet. If CoNet protocols are implemented in current border gateways (i.e. gateways running BGP), CSSs coincide with current Autonomous Systems. If CoNet protocols are implemented in all current routers, then CSSs coincide with current IP subnets. If CoNet protocols are implemented in nodes that interconnect different layer 2 networks, removing IP, CSSs coincide with these layer 2 networks.

CoNet nodes exchange *CoNet Information Units (CIUs)*: *interest CIUs* convey requests for named-data; *named-data CIUs* transport chunks of named-data, e.g., parts of a file (see Figure 31). To best fit the transfer units of an under-CoNet technology, all CIUs are carried in smaller CoNet data units named *carrier-packets*.

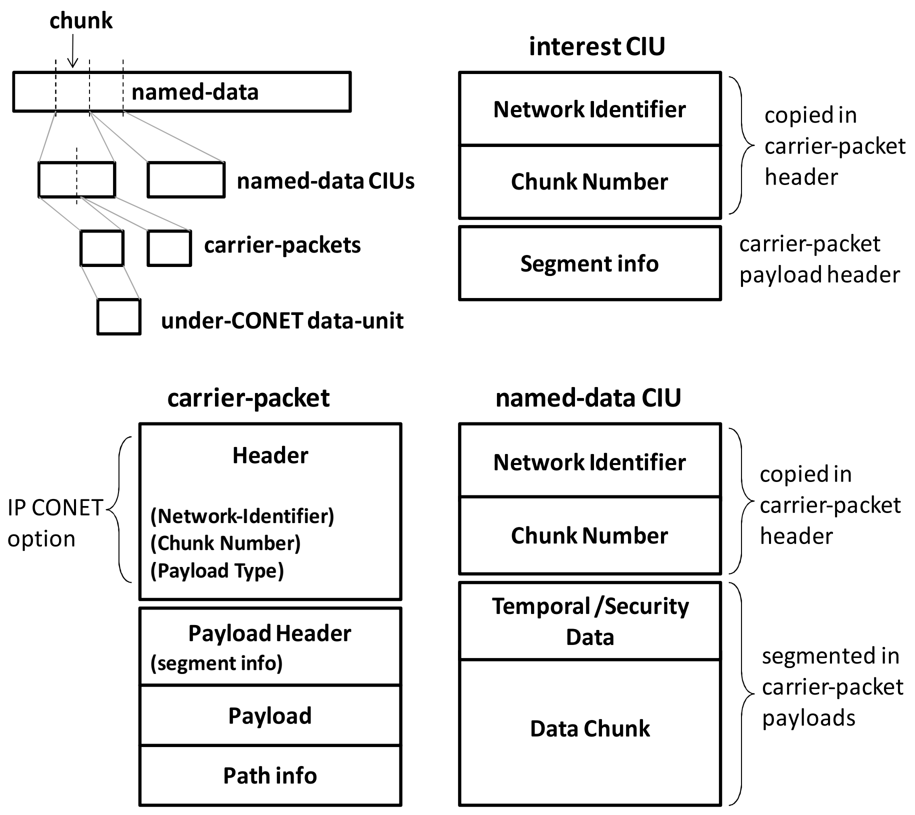


Figure 31: CoNet Information Units (CIUs) and carrier-packets

CoNet nodes are logically classified as end-nodes (ENs), serving-nodes (SNs), border-nodes (BNs), internal-nodes (INs) and name-system-nodes (NSNs). *End-nodes* are user devices that request named-data by issuing interest CIUs. *Serving-nodes* store, advertise and provide named-data by splitting the related sequence of bytes into one or more named-data CIUs, which are transferred by means of carrier-packets (see Figure 31). Border-*nodes,* located at the border between CSSs, forward carrier-packets by using CoNet routing mechanisms (i.e. routing-by-name and inter-CSS source-routing, as described below) and cache named-data CIUs. Optional *Internal-Nodes* can be deployed *inside* a CSS to provide in-network caches; unlike border-nodes, internal-nodes only use under-CoNet routing mechanisms to forward carrier-packets. The CSS uses optional *Name-System-Nodes* to assist the CoNet routing-by-name process. CoNet may be deployed following three approaches:

- *overlay approach*: CoNet runs on top of the IP layer; CSSs are couples of nodes connected by overlay links, e.g. UDP/IP tunnels, as in CSS n.1 in Figure 30;
- *clean slate approach*: CoNet runs on top of layer-2 technologies (e.g. Ethernet, PPP, MPLS LSP); CSSs are nodes connected by layer-2 links/networks, and CoNet replaces the IP layer, as in CSS n.3 in Figure 30;

- *integration approach*: CoNet functionality is integrated in the IP layer by means of a novel IPv4 option [6] or by means of an IPv6 extension header, as in CSS n.2 in Figure 30.

Readers are asked to note the three approaches are not mutually exclusive, but can be combined. Variants of the clean-slate and overlay approaches have been already discussed in the literature and by other research projects [7], [8], [9]. However, to the knowledge of the authors, the proposed integration approach is novel.

7.1.2 Model of operations

In this section we describe an example of the operation of CoNet in the scenario depicted in Figure 30, in which an end-node retrieves a chunk of named-resource from a serving-node. The example assumes that the serving-node has already advertised the related network-identifier in the CoNet by using a name-based routing protocol, as described in section 7.1.4 below. The retrieval of named-data involves a sequence of a *request* - *delivery* phases in which the end-node requests and obtains named-data CIUs and then reassembles the whole named-data (Figure 31). For simplicity, we consider a case in which the named-data are fully contained in a single named-data CIU that, in turn, is fully contained in a single carrier-packet. Therefore, only one request-delivery phase is needed.

Request

- An end-node requests the named-data CIU by issuing an interest CIU, which includes the network-identifier of the named-data; the interest CIU is encapsulated in a carrier-packet, named *I*.

- The end-node and intermediate border-nodes *route-by-name* the packet *I*. The route-by-name process singles out the *CSS address*[12] of the next border-node towards the serving-node, on the basis of the network-identifier contained in *I*. Then, the routing engine encapsulates the carrier-packet *I* in the under-CoNet data-unit and uses the CSS address as the destination address.

- The end-node and the set of traversed border nodes in the "upward" path record their CSS addresses in a control field of the carrier-packet *I* named *path-info*[13];

[12]A CSS address is an address consistent with the under-CoNet technology traversed by the packet (e.g. an IPv4 address).

[13] This info will be used to find the reverse-path to route the named-data CIU back to the requesting node, in the delivery phase. In [7] the same goal is achieved by maintaining states in network nodes. We are aware of the trade-offs involved, and we propose to use source-routing as we think that the number of CoNet border nodes traversed can be rather limited (e.g. CSSs could coincide with Internet Autonomous Systems). As an alternative, the path-info field could contain the NID of a named-sap, specifying where the requesting end-node can be reached. In this case, reverse-path routing could be performed by means of route-by-name procedures. This would be more convenient if CSSs were smaller and the number of traversed CoNet border-nodes higher. It would also give the network operator more freedom to choose the reverse-path.

- The internal-nodes parse carrier-packet *I* and then forward it by using the under-CoNet routing engine.

Delivery

- The first in-path CoNet node (BN, IN or SN), which is able to provide the named-data CIU requested by *I*, sends back the CIU, without further propagating *I*.
- The named-data CIU is encapsulated in a carrier-packet, named *C*. The carrier-packet *C* traverses the same CSSs as the carrier-packet *I*, but in the downward direction until it reaches the requesting end-node.
- The serving and the border nodes perform *inter-CSS* reverse-path routing in a *source-routing* fashion, using the path-info control field. This path-info is the copy of the path-info set up in *I* during upward routing.
- Within a CSS, the under-CoNet technology (e.g. IP) routes carrier-packet *C*. In these conditions, we can use traditional traffic engineering mechanisms.
- All border-nodes and internal-nodes in the downward path can cache the named-data CIU contained in *C*.

We observe that the use of inter-CSS source-routing on the reverse-path does not require “pending” states in the nodes traversed by the packed. We also observe that in end-to-end sessions bounded within the same IPv4 CSS, the path-info field is not necessary, as its only content would be the IP address of the end-node, already contained in the IP header.

7.1.3 CoNet protocol stack

As shown in Figure 32, every CoNet node has the CoNet and the *Under-CoNet* layers. The CoNet layer is connectionless, handles CIUs and carriers-packets, and provides other functionality (e.g. caching, security, etc.).

The end-nodes also have transport-level functionality, support reliability and flow control, and provide an Applications Programming Interface (API) (see section CoNet TE and CoNet TE API for the definition of the API between CoNet and upper layers). In what follows, we adopt the receiver-driven TCP-like approach proposed in [7], adapting it to our own terminology.

In this approach, the transport algorithm issues a sequence of interest CIUs and each of them requests only a small part of a named-data CIU, e.g. 1500 bytes per interest CIU. By controlling the sending rate of these interest CIUs, it is possible to obtain a TCP-like flow control mechanism. For instance, we could replace current TCP ACKs with interest CIUs and apply TCP congestion-window concepts to in-flight interest CIUs.

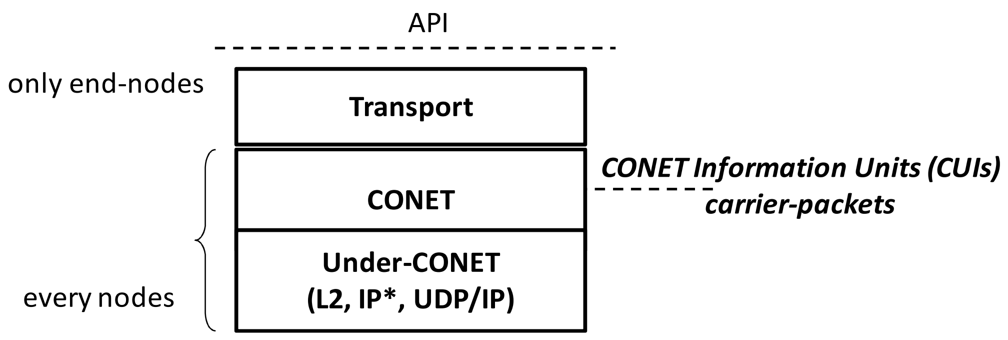


Figure 32: CoNet Protocol Stack

Figure 31 shows the packetization process, the CoNet CIUs (interest and named-data) and carrier-packets. In terms of notation, our interest CIU and named-data CIU correspond to the “interest packets” and “data packets” proposed in [7]. However the protocol information is different with respect to [7] (for example, our interest CIU carry the segment info field which is not present in [7]). In addition, we introduce the concept of carrier-packets, designed to improve CoNet forwarding speed.

Details of CoNet protocol operations are provided in deliverable D5.1 [10], which also describes the solution used when the CSS is an IP network, e.g. the IPv4 network in Figure 30.

The named-data (i.e. the representation in bytes of a resource) are split into different chunks. The optimal chunk size is the result of several tradeoffs. We favour a size roughly equivalent to the size of chunks in current P2P systems, e.g. 256-512 kbytes. However, the CoNet architecture can support variable chunk sizes.

Each chunk is inserted in a named-data CIU. Named-data CIUs are the data-units of the caching process and their control information includes the network-identifier, the chunk number, as well as timing and security data.

The network-identifier is a tuple <*namespace ID*, *name*>. The *namespace ID* determines the format of the *name* field. Thus, the *name* field is a namespace-specific string. Each namespace uses its own rules to generate unique *names* with its own format. We specify a default naming format, where the *name* is the composition of two hash values, i.e. *name*=<*hash (Principal identifier)*, *hash (Label)*>. Principal identifier and label are flat-names and a hash function transforms them to a fixed number of bytes (e.g., 6 bytes). A principal is the owner of her named-data and uses a *Principal identifier* whose hash is unique in the namespace. *Label* is an identifier chosen by the principal to uniquely differentiate her named-data. For instance, we could define the namespace “www” for the names of web resources (defined using current naming conventions), use the domain name (e.g. www.cnn.com) as the principal identifier and the URL (e.g. /foo/index.html) as the label.

Timing-data includes time information, like the expiry date, which can be exploited to implement digital forgetting mechanisms. Security-data make it possible to validate a named-data CIU before caching it or delivering it.

An interest CIU is a request for a set of bytes belonging to a named-data CIU, e.g., all bytes from byte X to byte Y (segment info field) of the named-data CIU n. Z (chunk number field).

Carrier-packets are low-level carriers of CIUs and are the data-units of the forwarding process. Carrier-packets are reassembled in border-nodes or in internal-nodes that want to cache the related named-data CIU, and in end-nodes; this operation is necessary to validate the content. Given that a named-data CIU could be too large to be transported by a single under-CoNet data-unit (e.g. 1.5kB for Ethernet and 64kB for IP) we introduce the concept of carrier-packets. Carrier-packets also make it possible to perform source-routing; carrier-packets are tightly associated with specific communication sessions between an end-node and a serving-node (or a cache).

7.1.4 Name-based routing: lookup-and-cache

The name-based routing is the mechanism used to update CoNet name-based routing tables. These are used by end-nodes or border-nodes to route-by-name interest CIUs. An entry in the name-based routing table contains the tuple <network-identifier, mask, next-hop, output-interface>. This is like an IP routing table entry. However, instead of net-prefixes we have *name-prefixes*, i.e. couples <network-identifier, mask>. Next-hop is the *CSS address* of the next border-node toward the serving-node.

In [11][7] the authors propose the use of traditional routing protocols, e.g. BGP or OSPF, to disseminate name-prefixes. We call this approach *prefix-dissemination*. The disadvantage of this system is that aggregation of names (i.e., network-identifiers) is not effective, unless names include information about the location of the serving node [12]. As a result, *prefix-dissemination* is likely to produce large name-based routing tables. As an example, consider supporting DNS domain names (as we would like to). In principle, it could be possible to perform limited location-based aggregation using top level domains [7]. However, this would not work for generic top level domains (.com, .net, etc.) which include very large numbers of names (the .com domain currently has about 90 million) in many different locations. Even in the case of country-code top level domains aggregation is unlikely to be very efficient. For instance, about 30% of .it names are outside Italy. Given that it is not feasible to include all possible names in the routing table, we propose a name-based routing, which we name *lookup-and-cache*. In this approach, a CoNet node (end-node or border-node) uses a fixed number of rows in its name-based routing table as a *route cache*. When a node misses the routing info required to route-by-name an interest CIU, it looks up its routing entry in a DNS like *name-system* and inserts the new entry in the route cache. When all rows are filled in, new routing entries replace old ones according to a suitable policy. From a logical point of view, a name-system serves a single CSS and a single namespace.

If a serving-node belongs to the same CSS as the node requesting the routing info, the name-system returns the CSS-address of the serving-node. If the serving-node is outside that CSS, the name-system returns the CSS-address of the egress border-node. If there are more than one serving-nodes, or egress border-nodes (due to replication operations), the name-system uses known techniques [14] to select the most convenient destination.

Prefix-dissemination and lookup-and-cache approaches can work separately or they can be combined, e.g. by using prefix-dissemination for the most popular named-data and lookup-and-cache for less popular data.

7.1.5 Integrating CoNet in IP

In this section, we describe a technique to support the CoNet in a CSS that is an IP network (IP-CSS), e.g. CSS n.2 in Figure 30. The IP network could correspond to the whole public Internet. Our approach thus provides a way to offer CoNet services over the Internet. To achieve this goal we propose what we can call an *integration approach.* This approach i) does not require us to give up IP, as in the clean-state approach; ii) performs better than a CoNet on top of IP, as in the overlay approach. The idea of the integration approach is to *make IP* itself *content-aware*. We propose to transport the header of a CoNet carrier-packet in a novel IPv4 option (or IPv6 extension header), which we call the CoNet option (see Figure 31 and [6]). In this kind of IP-CSS, border and internal CoNet nodes are simply IP routers extended with CoNet functionality.

Figure 33 shows a possible architecture for a border or internal CoNet node. A *fast forwarding path* handles forwarding operations for CoNet carrier-packets and for plain IP packets. The hardware (RIB or FIB) routing table does not only include IP net-prefixes as entries but also name-prefixes. A name-prefix entry may refer both to remote named-data and to local cached named-data. In the latter case the routing entry points to the local cache engine, therefore the CoNet node will return the locally cached named-data when receiving an Interest CIU that matches the routing entry. Other CoNet and IP functions with less stringent delay constraints are performed by a CPU. Examples include IP and name-based routing, execution of caching algorithms, reassembly of named-data CIUs for caching, replies to interest CIUs requesting cached data, etc.

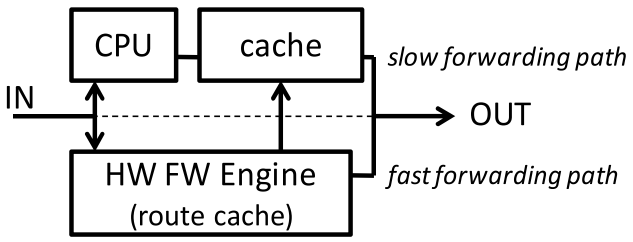


Figure 33: Architecture of a CoNet node of an IP-CSS

Most of these operations require parsing of incoming CoNet CIUs. With our approach, it is possible to forward an incoming CIU using the HW engine (without affecting forwarding performance) and in parallel to process it in the CPU in the so called “slow-path”.

The advantages of this approach with respect to the overlay approach is that it allows CoNet nodes to quickly forward carrier-packets, without having to perform a slow deep packet inspection. This is a fundamental requirement for the implementation of content-centric features in nodes where a high packet rate demands fast forwarding. In addition, the integration approach makes it possible to restrict deployment of CoNet routing-by-name functions to a subset of nodes (i.e. border-nodes and end-nodes) while allowing caching in all nodes running the new IP option (i.e. internal nodes). By contrast, in the overlay approach, the only way to implement caching in all nodes is to deploy routing-by-name functionality in all nodes.

The disadvantage of the integration approach is that it requires a new IP option. However this is far less disruptive than the clean-state approach. The approach lends itself to different deployment scenarios. In an extreme case the IP-CSS could be the whole Internet and routing-by-name functions would be performed only in end-nodes. All that would be required to implement in-network caching would be to introduce the new IP option; there would be no need to introduce routing-by-name functions in routers.

In another scenario we could partition the Internet into a set of IPv4 CSSs that interoperate exclusively through CoNet protocols. In this scenario, each CSS would uses an IPv4 addressing scheme unique to the CSS. IP routing would thus be restricted to individual CSS. This would allow new providers to offer public CoNet services without using public IP addresses, and without increasing the size of the routing tables of Internet backbone routers – a critical scalability issue for all CCN architectures and even for the current Internet [12].

7.1.6 CoNet Application Program Interface

This section describes the CoNet Application Programming Interface provided by the CoNet. ***Table 9*** reports a preliminary set of primitives. Thanks to these primitives, the CoNet allows its users to:

- Advertise named-resources, i.e. resources identified by a network-identifier (NID);
- Access named-resources;
- Deliver named-data and un-named-data (i.e., upper layer data).

The scope of applicability of some primitives depends on the type of the named-resource they deal with. We thus distinguish two classes of named-resources: named-data (i.e., documents, video, images, structured information, VDIs, etc.) and named-service-access-points.

For examples of the use of these primitives see the following section.

Table 9 – Service primitives offered by the CoNet API

Service Primitive	Applicability	Origin	Output	Description
Advertise (NID, resource-type, [resource /port], expiry, auth,…)	Named-data and named-service-access-points	CoNet user (e.g., application or middleware)	success /failure	This primitive is used by a CoNet user to make a <u>local</u> named-resource accessible to other CoNet users. NID is the network-identifier of the named-resource. resource-type specifies if the resource is named-data or a named-service-access-point. In the case of named-data, resource is actual content (e.g., a document, an image, media, file, a VDI). In the case of a named-service-access-point, port is a *local entity port* where an entity expects to receive data through an Indication primitive (this corresponds to the transport port that IP networks use as a local address in a host to deliver data to a given entity). expiry defines a date after which the advertisement will be automatically revoked. auth is a set of information that authenticates the user advertising a named-resource with a specific NID. This prevents the forging of NIDs. Failures may occur if Auth is not valid, a resource type is not consistent, etc. After Advertise, a new named resource is introduced into the network. The resource will be reachable by name by any entity in the CONVERGENCE network. Given that CoNet supports routes-by-NID, this information has to be introduced into the network to route requests for the new named resource towards the appropriate network nodes Thus, Advertise changes routing information on CoNet.
Update (NID, resource-type, resource/port, expiry, auth,…)	Named-data and named-service-access-points	CoNet user (e.g., application or middleware)	success /failure	This primitive is used by a CoNet user to update the resource associated with a previously advertised NID. The meaning of the parameters is the same as in Advertise. However, in this case it is not necessary to introduce or modify routing information.

Revoke (NID, auth,…)	Named-data and named-service-access-points	CoNet user	success /failure	This primitive is used by a CoNet user to revoke (and delete) a local named-resource, identified by NID. auth is a set of information that authenticates the user seeking to revoke the named-resource. A failure occurs when the named-resource is not available. Like Advertise ,Revoke modifies CoNet routing information
Get (NID)	Named-Data	CoNet user	data / failure	This primitive is used by a CoNet user to retrieve a named-resource identified by NID. A failure occurs if the named-resource is not available.
Send2Name (NID, data, port)	Named-service-access-points	CoNet user	success / failure	This primitive is used by a CoNet user to send un-named-data (i.e. upper layer data) toward a named-service-access-point, identified by NID. If the CoNet user waits for a response from the remote endpoint, the parameter port is the local entity port where the CoNet user expects the Indication primitive.
Indication (LID, data)	Service access point	CoNet		When un-named-data (i.e. upper layer data) are received by a service-access-point, this primitive is invoked by the CoNet on the entity which is listening to the local entity port associated with the service-access-point (see advertise). The primitive gives the un-named-data and the temporary identifier LID of the location of the remote entity that has sent these un-named-data to the service entity listening to the port.
Send2Location (LID, data, port)	Service access point	CoNet user		This primitive is used by a CoNet user to send un-named-data (i.e. upper layer data), via unicast, to a specific entity whose location is temporary identified by LID. If the CoNet user expects a response from the remote endpoint, the parameter port is the local entity port where the CoNet user expects the Indication primitive.

7.1.7 Examples of use of the CoNet Application Program Interface

In this section, we show how the CoNet API could be exploited to support typical operations. Details are preliminary. The final specifications will be completed at a later stage in our work.

7.1.7.1 Advertising local data

Figure 34 depicts the case of a CoNet user (e.g. a provider application), who wishes to make available to other CoNet users the local file "times09-30-10.pdf", with network-identifier (NID) "the times: Sept-30-2010". The provider calls **Advertise**. The local CoNet node stores the file locally, updates an internal database that maps "the times: Sept-30-2010" to the file "times09-30-10.pdf," and updates the routing plane of the whole CoNet, so that the local CoNet node becomes a provider of the named-resource "the times: Sept-30-2010". There may also be other providers.

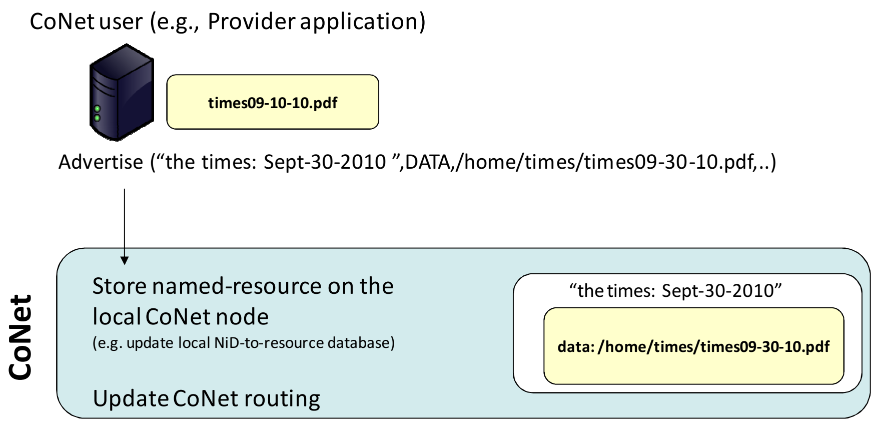
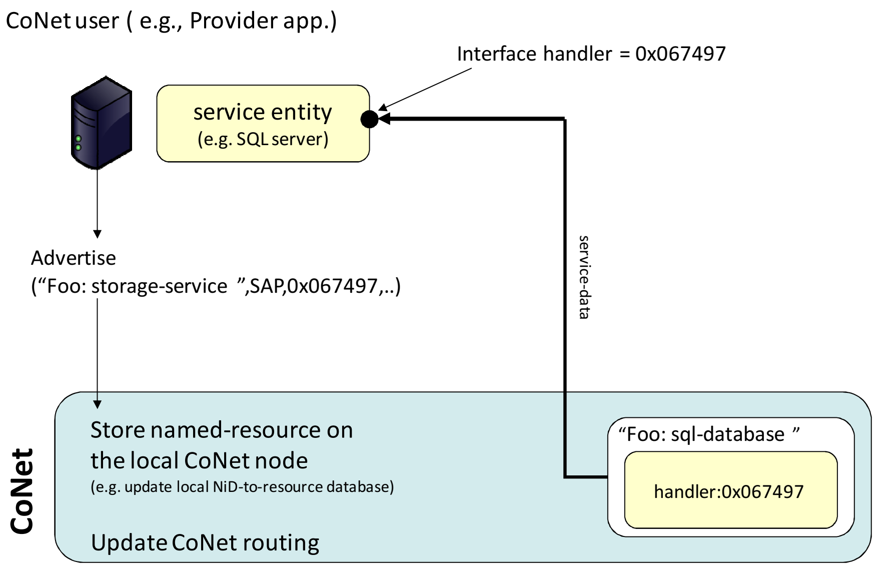

CoNet user (e.g., Provider application)
times09-10-10.pdf
Advertise ("the times: Sept-30-2010 ",DATA,/home/times/times09-30-10.pdf,..)
CoNet
Store named-resource on the local CoNet node
(e.g. update local NiD-to-resource database)
Update CoNet routing
"the times: Sept-30-2010"
data: /home/times/times09-30-10.pdf

Figure 34: Advertising a local named-resource (named-data)

7.1.7.2 Advertising local services

Figure 35 depicts the case of a CoNet user (e.g. a provider application) who wishes to make available to other CoNet users a local SQL database service with network-identifier (NID) "Foo: sql-database". The provider runs the relevant server application, which is configured to receive its service-data (un-named-data for the CoNet) through the local entity port 0x067497. Then the provider application uses the Advertise primitive. Subsequently, the local CoNet node updates an internal database that maps "Foo: sql-database" to the local entity port 0x067497 and updates the routing plane of the whole CoNet, so that the local CoNet node becomes a provider of the named-resource "Foo: sql-database". To improve the reliability of database access, the Foo company may replicate the database by repeating the same advertising operation on different CoNet nodes.



CoNet user (e.g., Provider app.)
Interface handler = 0x067497
service entity
(e.g. SQL server)
Advertise
("Foo: storage-service ",SAP,0x067497,..)
service-data
CoNet
Store named-resource on the local CoNet node
(e.g. update local NiD-to-resource database)
Update CoNet routing
"Foo: sql-database "
handler:0x067497

Figure 35: Advertising a local named-resource (named-service-access-point)

7.1.7.3 Downloading data

Figure 36 depicts the case of a CoNet user (e.g. a end-user application) who wishes to download the document identified by the NID "the times: Sept-30-2010". The user application uses the GET primitive. The local CoNet node identifies the specific instance of the GET primitive with a Location Identifier (LID). Then the CoNet node sends the network data unit containing (at least) the LID and the NID. The CoNet *routes-by-NID* this data unit to the best remote node that is advertising "the times: Sept-30-10". When the remote node is reached, the CoNet functionality directly replies by sending-back the data-unit, containing the file "times09-10-10.pdf". This data-unit is routed using the LID and contains the NID ("the times: Sept-30-10"). The presence of the NID in the reply allows intermediate nodes to identify the traversing content and possibly to perform caching.

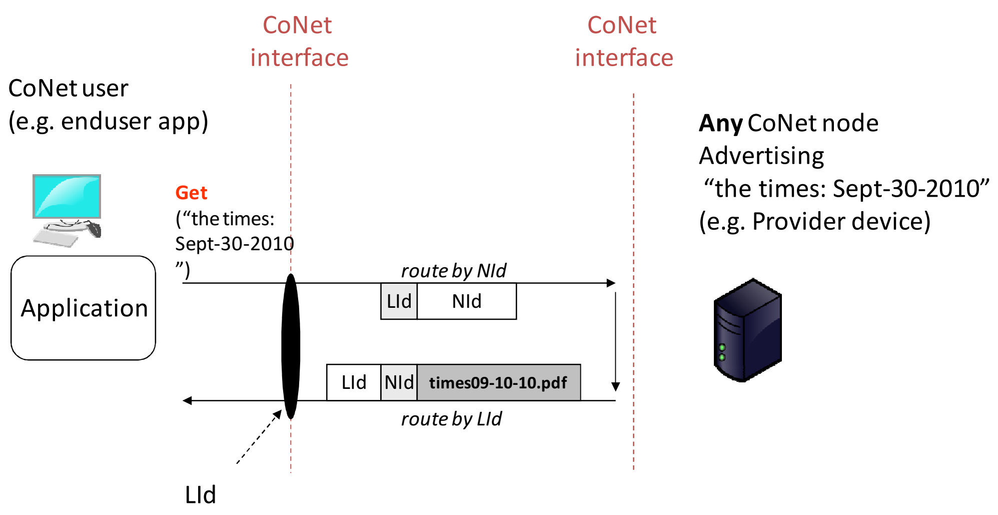


Figure 36: Downloading a named-resource (named-data)

7.1.7.4 Request-response service session

Figure 37 depicts the case of a CoNet user (e.g. an end-user client application), who wishes to send *un-named-data* (e.g. a SQL query)) to a remote server with NID "Foo: sql-database". The user application uses Send2Name, specifying the local entity port 0x0076 where the response is expected. The local CoNet node associates the local entity port with a Location Identifier, LID. Then the CoNet node sends the network data unit-containing the LID and the NID. The CoNet routes-by-NID the data-unit to the best remote node that is advertising "Foo: sql-database". When the remote node is reached, the remote CoNet functionality invokes Indication on the entity listening to the local entity port of the server application, and transfers the LID and the un-named-data, i.e. the SQL query. Subsequently, the server processes the request and replies to the client using Send2Location and including the response (un-named-data), the client LID and the local entity port 0x067497 of the local interface where further replies from the client are expected. The CoNet adds a local-identifier LID* addressing the local entity port and routes by LID the resulting data-units. When the *un-named-data* reach the client, the CoNet functionality of the client device sends an Indication to the client application, which includes both the server response and the LID* of the remote server.

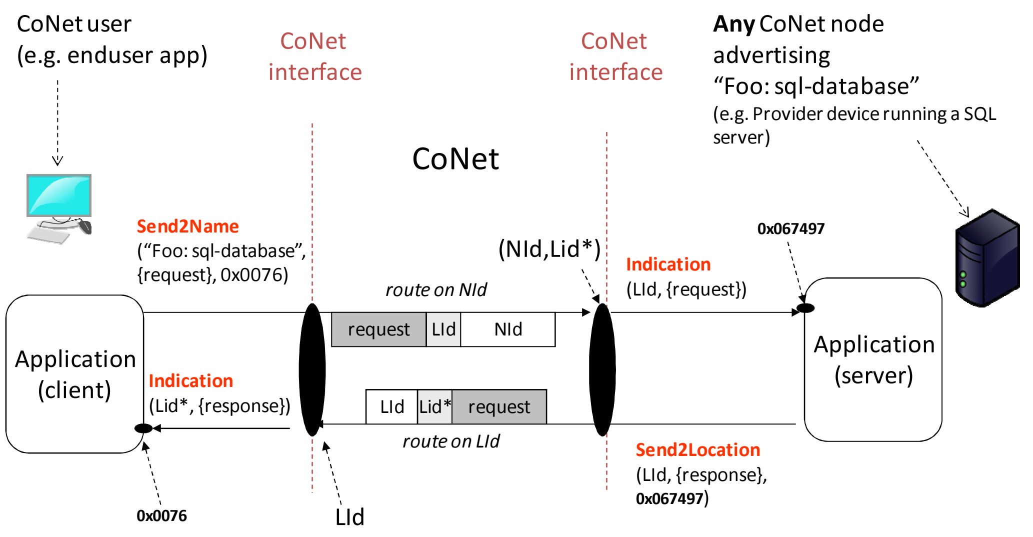


Figure 37: Request-response service session

7.1.7.5 Interactive service session

Figure 38 depicts the case of a CoNet user (e.g. an end-user CoMid entity) who wishes to interactively exchange un-named-data (i.e., middleware messages) with a remote CoMid entity identified by the NID "Foo: mxm-content-provider". This case is equivalent to the case of the request-response service session described in Section 7.1.7.4. However, after the first interaction, the endpoints continue to exchange un-named-data using Send2Location.

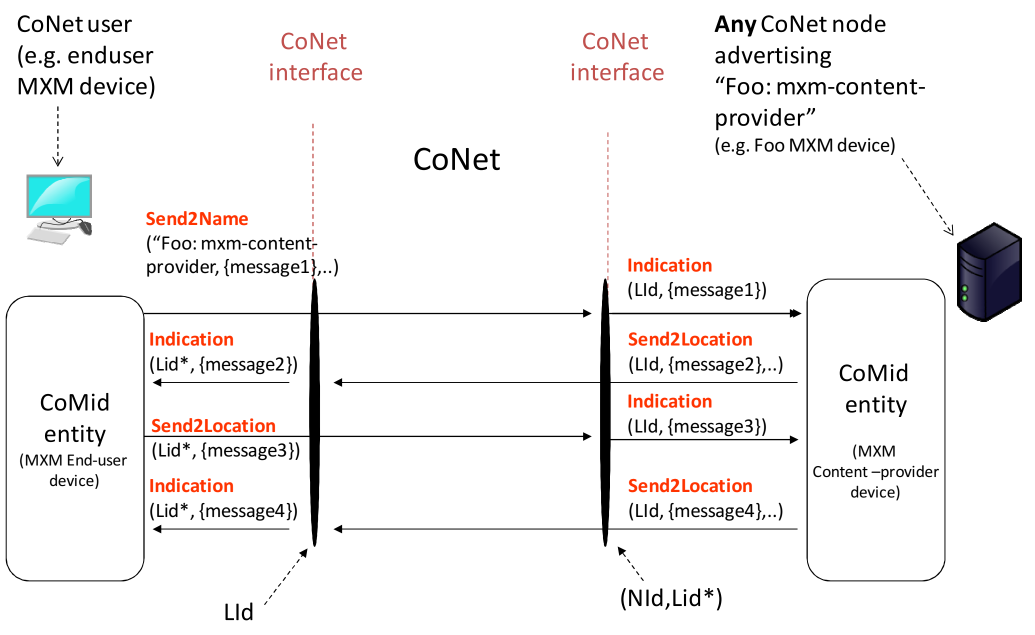


Figure 38: Interactive service-session

7.1.7.6 Remote advertising

The advertising examples presented in sections 7.1.7.1 and 7.1.7.2 describe cases in which the named-resource is provided locally. This means that a user who wishes to provide a named-resource should use a device that is always connected to the network. However, the user may not want to manage such a device and may prefer to use a hosting provider who will take care of advertising her named resources. This can be achieved by using an application service, rather than a CoNet service. Thus, a Foo hosting provider may advertise a service named "Foo: hosting-service" to which the user can "tunnel" her named-resource (i.e., the named-data and the relevant NID). The hosting-service will then advertise (and store) the named-resource locally.

7.2 Security Component (CoSec)

7.2.1 Survey

In the architectural design of CONVERGENCE, CoSec is the component within the Computing Platform level responsible for handling cryptographic protocols and security related tasks. The CoSec has a *distributed architecture* encompassing several independent and possibly distant components with each component comprising software as well as hardware. In the application flow of security protocols, these components interact with each other. As a consequence, their APIs can be quite complex.

CoSec offers some "high-level" security support to CoMid; for instance encryption of content, authentication of users, etc. (Note that it can offer similar support to CoNet). Within CoMid, these services are offered to all CoMid engines by the Security TE.

In its current state, the MPEG API for the Security TE is not fully aligned with this architecture. There are two reasons for this. First, the MPEG API offers a variety of low-level cryptographic functionality (such as key-generation, random-number generation) which should better be considered "low-level" cryptographic algorithms, to be allocated to CoSec internal behaviour. Second, in CoSec, the API needs to be significantly extended, especially when the secure repository is a smart card. In fact, a set of functionality (e.g. user authentication, key unwrapping) selected for CONVERGENCE can be realized involving smart cards, since the smart card will be the preferred choice for a secure repository in CONVERGENCE. It is considered a security enhancement that some security relevant functions are even entirely performed on-card.

We therefore plan to reconsider the API for the current Security TE, and work on a new API for CoSec service offered to TEs and a second more complex API for CoSec internal cryptographic functions. The full definition of these API lies beyond the scope of D3.2 and will be completed in subsequent work (see also beginning of Section 9.11 and 9.11.2). The current Security TE API is reported in the ANNEX, documenting the actual status of our work and starting point for next work.

CoSec components are **distributed on the following computing platforms**:

1. Client computers (e.g. end-user laptops)
2. Application servers
3. Peers (e.g. node computers in a network)
4. Smart Cards (typically held by the end-users)

Typically, *several* of the above components will interact in the execution of *cryptographic protocols*. Due to the nature of these protocols and their underlying parameters, the interface structure *between* CoSec functions will inevitably be complex. A detailed description lies beyond the scope of this document.

7.2.2 Cryptographic Primitives

CoSec will use established *off-the-shelf* solutions for cryptographic functionalities.

1. **Fast *symmetric encryption* and *decryption* of content** (F.ENG-VDI c.)
   a. Established primitives like AES-CBC
   b. Key derivation through sophisticated protocols like ABE (Attribute Based Encryption),, IBE (Identity Based Encryption).
2. ***Asymmetric cryptography*** (F.USER-AUTH-CONV, F.USER-AUTH-SERV, F.SERV-AUTH, F.PEER-AUTH, F.ENG-AUTH, F.ENG-VDI,F.ENG-VDI-SIGN-THPART)

   a. Established primitives like RSA or Elliptic Curves
   b. Used for key agreements, signatures, certificates, etc.
3. **Basic primitives like cryptographic hashes** (F.USER-AUTH-CONV, F.USER-AUTH-SERV, F.SERV-AUTH, F.PEER-AUTH, F.ENG-AUTH, F.ENG-VDI)

Beyond these standard solutions, to enable the fulfilment of the Security Functional Requirements the following much more *advanced primitives* shall be used:

4. ***Group Signature Protocol*** (F.USER-ID b., F-USER-ID-GROUP)
   a. Sign a VDI's content anonymously on behalf of a pre-specified group
   b. Allow unveiling anonymity only on request
5. ***Identity and Attribute Based Encryption*** (F.TE-LIC-ISS, F.TE-LIC-ENF)
   a. Recipients of content (or messages) are assigned specific (arbitrary) attributes
   b. The provider of content can encrypt according to attributes, so that the possession of the same attributes is needed to decrypt the content
6. ***Pseudonymous access via "Restricted Identification".*** (F.USER-ID c., F-USER-ID-PSEUDONYM)
   The use of pseudonymous access is motivated by data protection issues.
   a. Within a specific context, each user has a unique pseudonym to identify himself
   b. For disjoint context, users' pseudonyms cannot be linked

For the benefit of the reader, a short overview of each of these more sophisticated schemes is given below:

7.2.2.1 Group Signature Protocol

Group Signature Schemes allow all members of a previously created group to issue a digital signature on behalf of the entire group. A third party can use the group's public key to verify if the signature originates from the group (by using one common public key associated to that group), but cannot ascertain the identity of the group member who made the signature. Multiple anonymous signatures issued by the same member of the group cannot be linked. (This implies that each signer is *anonymous* and not just *pseudonymous*).

The group is administrated by a" group master", who must be absolutely trustworthy. The group master holds a "master key" with which he can register new members, revoke existing members, and unveil the identity of a signer in the event of a dispute. This ability may be a procedural requirement in some applications, where unconditional anonymity may not be permitted. Note that Group Signatures are quite a recent development and legal implications/requirements related to their use are not clearly defined.

In the LMU scenario, for example, the group consists of all registered students who have access to (some of) the lectures offered, and all lecturers. Membership requires a *registration* process. During registration each student receives a smart card that includes a personal signature key (used to issue personal digital signatures), but also a private group signature key that enables the student to issue an *anonymous* group signature on behalf of the entire student group. An exemplary usage of such a group signature consists in a student's ability to sign annotations (e.g. on behalf of a group attending a lecture) without revealing his identity.

7.2.2.2 Identity Based Encryption – Attribute Based Encryption

In recent years, research on Identity Based Encryption (IBE) and Attribute Based Encryption (ABE) have made significant progress, and cryptographic protocols based on these schemes have matured to a degree in which they can be deployed in real life applications.

IBE allows asymmetric cryptography (encryption and signatures) without the burden of certificate administration typically associated with the creation of a Public Key Infrastructure (PKI). In CONVERGENCE, VDIs can be encrypted and decrypted without the need to establish a cumbersome PKI.

Attribute Based Encryption makes it possible to map complex *licenses* to *attributes*. The basic idea of an Attribute Based Encryption scheme is to partially implement license enforcement by encrypting content (or more precisely a content key) according to determined attributes. In this way, only key holders with the attributes required by the encrypted content will be able to decrypt it. By contrast, conventional license checking can be bypassed by a fraudulent user who tampers with hardware and software components.

7.2.2.3 Pseudonymous Access – "Restricted Identification"

In the context of this report, we use Pseudonymous access to refer to the specific "*Restricted Identification*" technology used in the German Electronic Passport. The technology allows each registered user to derive pseudo-identities from a personal private key contained in his smart card. The same private key can be used to derive *"sector"-specific* pseudonyms. This means that an individual user's pseudonym will always be the same in each individual sector, while his or her pseudonyms for disjoint sectors will be different and cannot be linked.

Sectors need not restrict to local areas, but can be fields of applications, different online-providers of services, and so on.

The method also allows revocation and blacklisting of pseudonyms.

We propose to use this form of pseudonymous for the CONVERGENCE smart retailing trial. The main motivation for using it is to introduce data protection and improve privacy for users of the system. Customers can use their pseudonyms to order goods without revealing their true identity. Different shops will correspond to different sectors. This will prevent shops from cross-linking their data and prevent third parties from mining data from different shops to profile individual customers.

7.2.3 Smart Cards in CONVERGENCE

Cryptographic software, or at least essential part of it, should run in a trustworthy environment. Naturally, the easiest assumption is to regard the entire network as safe, and to assume that registered devices (laptops, PCs, etc.), once checked, will remain secure devices and can be trusted. Unfortunately, the opposite is true. A network is anything but secure, and devices (even if initially correct) can be tampered with by fraudulent users, or manipulated by third parties.

Many of these problems can be solved, or at least mitigated, by the use of *smart cards* as *secure hardware modules*. Smart card software is almost completely safe against tampering. This means that an attacker will find it very hard to access confidential content stored or processed on a well-designed system. In this way, the smart card serves as a reliable outpost for the service provider, which continues to control it, even though it is permanently in the hands of an end-user.

On the other hand, smart cards can also contain secret information which is not available anywhere else, not even on the manufacturer's or card issuer's site. A typical example is a signature key: such keys are generated *on-card*, are *unknown* to *any* outside entity, and always remain in the physical possession of the card holder. The card holder can therefore be certain that no one can forge her signature without physical access to her smart card - not even the service provider or card manufacturer can do this.

7.2.4 Cryptographic Protocols for specific Security Functional Requirements

7.2.4.1 Key generation – User Registration

Membership in the CONVERGENCE network scenarios requires a registration process. Registration is administrated by an *Identity Provider*, and comprises *identification* and cryptographic *key generation*.

End-users will be equipped with smart cards containing confidential keys. On registration, each end-user receives a *smart card* containing at least the following key sets.

A unique *signature key* pair

The key shall be generated *on the smart card* itself, *during registration*, according to parameters agreed upon by CONVERGENCE. The secret part of the signature key shall be stored in such a way that it can *never* be read out of the smart card by any entity whatsoever.

The key shall be *certified* during registration[14]; the certificate (containing the public key part) shall be signed by the registration authority and stored on the smart card, as well as on a CONVERGENCE server.

The generation of this key pair will be *mandatory* for each member wishing to use a smart card as secure repository.

A unique *encryption key* pair

A private-public key pair shall be assigned to each user, generated according to the same policy chosen for the signature key generation. However, the encryption key pair must not be identical to the signature key pair (it is not allowed to use the same key pair), but shall be generated completely independent from the aforementioned signature key pair.

The key shall be certified during registration; and the certificate shall be stored on the member's smart card, as well as on CONVERGENCE servers.

We consider the generation of this key pair *mandatory* for each member wishing to use a smart card as secure repository.

A unique group signature key

During registration, each member's smart card shall generate a random seed, securely transmitted to the group administration authority for computation of the complementary secret key part. Both parts constitute the private key of a member, and shall be safely stored on his or her smart card.

Note that unlike the signature key and encryption key, we cannot insist that each member's private group signature key be known only to the smart card. Instead, the group signature scheme requires the existence of an absolutely trustworthy "group master" administrating the group with a master secret key; see section above.

A unique private key for Identity/Attribute Based Encryption

As for the group signature key above, Identity Based Encryption or Attribute Based Encryption requires the private key for each member to be derived from a "master" key held by a trusted authority. The setup is similar to the setup for the group signature scheme.

[14] Note that this requires a protocol allowing verification of the correctness and possession of the private key part without having access to it. A simple example of such a protocol consists in obtaining a signature over a random challenge of the verifiers choice under supervision.

7.2.5 Authentication of users

Authentication is required by F.USER-AUTH-CONV, F.USER-AUTH-SERV, F.SERV-AUTH, F.PEER-AUTH, F.ENG-VDI and can be achieved through:

a. Challenge-Response protocols involving signatures
b. Key-agreement protocols which do not involve end-user's signatures

7.2.6 Integrity/Authenticity

Integrity and authenticity are required by F.ENG-VDI, O.VDI-INT, O.VDI-AUTH, F.ENG-INT, F.ENG-AUTH and may be achieved through:

a. Signing
b. Use of hash-tables (e.g. containing identifier and hash-value) stored by a trustworthy service

7.2.7 Licensing

Licensing splits up into license issuance (F.TE-LIC-ISS) and license enforcement (F.TE-LIC-ENF). It may basically be achieved through:

a. License Technology Engine, using REL
b. On-card license validation, using Card Verifiable Certificates
c. Attribute Based Encryption, mapping license conditions to attributes

7.2.8 Example Protocols

7.2.8.1 First example: Smart Card authentication using Key Agreement Scheme

The smart card is equipped with a static asymmetric key pair. To achieve authentication, it proves possession of the secret key to a remote server.

Entities involved:

a. Remote peer
b. End-user's local client
c. End-user's Smart Card

Protocol Flow:

1. The smart card sends its static public key to the local client, along with a certificate. The local client transmits both of them to the remote peer.
2. The remote peer generates an ephemeral asymmetric key pair, and sends the public key part to the local client. The local client transmits it to the smart card.
3. Both the remote peer and the smart card compute a common secret key *K* through Diffie-Hellman key agreement.

4. The smart card chooses a random nonce, and generates an authentication token derived from this nonce through a MAC (Message Authentication Code) using *K*. The smart card sends both nonce and token to the local client, which transmits it to the remote peer.
5. The remote peer verifies the token by reproducing a MAC over the nonce with the same secret key *K*.

Note: *The benefit in using a smart card lies in the fact that the end-user's private key remains within the smart card. The end-user need only have limited trust in the local client machine (e.g. a PC in an internet cafe).*

7.2.8.2 Second example: A Service Application's authentication a Challenge-Response

The service application is equipped with a static asymmetric key pair. It authenticates towards an end-user's smart card by proving possession of the secret key.

Entities involved:

a. Service Application
b. End-user's local client
c. End-user's smart Card

Protocol Flow:

1. The service application sends a card-verifiable certificate (containing its public key) to the end-user's local client. – The local client transmits the certificate to the smart card for on-card validation.
2. The smart card chooses a random challenge and sends it to the local client, which transmits it to the application server.
3. The service application signs the challenge, and returns the signature to the local client, which transmits it to the smart card for verification
4. The smart card verifies the signature in order to authenticate the service application.

Note: *The benefit in using a smart card lies in the fact that the end-user's smart card can check the service application's certificate, and validate the authentication. On the one hand, the end-user need only have limited trust in the local client machine (e.g. a PC in an internet cafe). On the other hand, the issuer of the smart card (a Service Provider) can enforce authentication even in an environment where he may mistrust a user's local client (which may be the case if an end-user himself has an interest in manipulating his device).*

8 Bibliography

[1] Network layer solutions for a content-centric Internet. A. Detti, N. Blefari-Melazzi in "Trustworthy Internet", Springer 2010.

[2] A data-oriented (and beyond) network architecture. T. Koponen, M. Chawla, B.G. Chun, A. Ermolinskiy, K. H. Kim, S. Shenker, and I. Stoica in procs. of ACM SIGCOMM'07, 2007

[3] Networking named content. V. Jacobson, D. K. Smetters, et al., Fifth ACM International Conference on emerging Networking EXperiments and Technologies (CoNEXT), 2009

[4] On the Cache-and-Forward Network Architecture L. Dong, H. Liu, Y. Zhang, S. Paul, D. Raychudhuri in procs. of IEEE International Conference on Communications 2009, ICC 2009

[5] An Introduction to Petname Systems, Mark Steigler, available at http://www.skyhunter.com/marcs/petnames/IntroPetNames.html

[6] An IPv4 Option to support Content Networking, A. Detti, S. Salsano, N. Blefari-Melazzi, Internet Draft, draft-detti-conet-ip-option-00, Work in progress, March 2011.

[7] Networking named content, V. Jacobson, et al., in Proc. of ACM CoNEXT 2009

[8] PURSUIT project website: www.fp7-pursuit.eu

[9] 4WARD project website: www.4ward-project.eu

[10] CONVERGENCE Project Deliverable D5.1 "Requirements and Initial Protocol Architecture"

[11] TRIAD: a scalable deployable NAT-based internet architecture, D. Cheriton, M. Gritter, Technical Report (2000)"

[12] Report from the IAB Workshop on Routing and Addressing, D. Meyer, L. Zhang, K. Fall, RFC 4984

[13] D. Oran, "OSI IS-IS intra-domain routing protocol", IETF RFC 1142

[14] D. C. Verma "Content Distribution Networks", Wiley-Interscience

[15] T. Koponen, M. Chawla, B.G. Chun, et al.: "A data-oriented (and beyond) network architecture", ACM SIGCOMM 2007

[16] D. Smetters, V. Jacobson: "Securing Network Content", PARC technical report, October 2009

[17] K Katsaros, G. Xylomenos, G. C. Polyzos: "MultiCache: An overlay architecture for information-centric networking", Computer Networks, Elsevier, Volume 55, Issue 4, 10 March 2011, Pages 936-947

[18] S. Oueslati, J. Roberts, N. Sbihi: "Ideas on Traffic Management in CCN", Information-Centric Networking, Dagstuhl Seminar

[19] The Many Faces of Publish-Subscribe, P.T. Eugster, P.A. Felber, R. Guerraoui, A. Kermarrec, in ACM Computing Surveys (CSUR), Volume 35, Issue 2, 2003

[20] W3C HTML 4.01 Specification, Section 12 – Links, 1997.http://www.w3.org/TR/html401/struct/links.html
[21] ISO/IEC 23006 – Information Technology – Multimedia Service Platform Technologies (MPEG-M)
[22] ISO/IEC 21000-2 – Information technology -- Multimedia framework (MPEG-21) -- Part 2: Digital Item Declaration
[23] ISO/IEC 21000-3 – Information technology -- Multimedia framework (MPEG-21) -- Part 3: Digital Item Identification
[24] ISO/IEC 21000-4 – Information technology -- Multimedia framework (MPEG-21) -- Part 4: Intellectual Property Management and Protection Components
[25] ISO/IEC 21000-5 – Information technology -- Multimedia framework (MPEG-21) -- Part 5: Rights Expression Language
[26] ISO/IEC 21000-15 – Information technology -- Multimedia framework (MPEG-21) -- Part 15: Event Reporting
[27] ISO/IEC 21000-19 – Information technology -- Multimedia framework (MPEG-21) -- Part 19: Media Value Chain Ontology
[28] ISO/IEC 21000-20 – Information technology -- Multimedia framework (MPEG-21) -- Part 20:Contract Expression Language
[29] ISO/IEC 23006-1 – Information technology -- Multimedia Service Platform Technologies – Part 1 - Architecture
[30] ISO/IEC 23006-2 – Information technology -- Multimedia Service Platform Technologies – Part 2 – MPEG Extensible Middleware API
[31] ISO/IEC 23006-4 – Information technology -- Multimedia Service Platform Technologies – Part 4 – Elementary Services
[32] ISO/IEC 23006-5 – Information technology -- Multimedia Service Platform Technologies – Part 5 – Service Aggregation
[33] B. B. Mandelbrot, The Fractal Geometry of Nature, W.H. Freeman and Company, New York, 1982
[34] P. T. Eugster, R. Guerraoui, A.-M. Kermarrec, L. Massouli´e, Epidemic information dissemination in distributed systems, IEEE Computer 37 (2004) 60–67
[35] R. Friedman, D. Gavidia, L. Rodrigues, A. C. Viana, S. Voulgaris, Gossiping on manets: the beauty and the beast, SIGOPS Operating Systems Review 41 (2007) 67–74
[36] A. J. Ganesh, A. M. Kermarrec, L. Massoulié, Scamp: Peer-to-peer lightweight membership service for large-scale group communication, in: J. Crowcroft, M. Hofmann (Eds.), Networked Group Communication, volume 2233 of Lecture Notes in Computer Science, Springer Berlin /Heidelberg, 2001, pp. 44–55

[37] A. M. Kermarrec, L. Massoulié, A. J. Ganesh, Probabilistic reliable dissemination in large-scale systems, IEEE Transactions on Parallel and Distributed Systems 14 (2003) 248–258

[38] M. Muhr, R. Kern, M. Granitzer, Analysis of structural relationships for hierarchical cluster labeling, in: Proceeding of the 33rd international ACM SIGIR conference on Research and development in information retrieval, SIGIR ’10, ACM, New York, NY, USA, 2010, pp. 178–185

[39] Ahmed, R.; Boutaba, R.; "A Survey of Distributed Search Techniques in Large Scale Distributed Systems," Communications Surveys & Tutorials, IEEE , vol.13, no.2, pp.150-167, Second Quarter 2011

[40] S. Pantelopoulos, P. Gkonis: “Tools and Sample Applications Plans and Vision” CONVERGENCE public deliverable D7.1, July 2011.

[41] The IMDB Mapping Movie Ontology by Gunnar Grimnes, http://www.csd.abdn.ac.uk/~ggrimnes/dev/imdb/IMDB.rdfs

[42] National Retail Federation, http://www.nrf.com/

9 ANNEX A - APIs for CoMid Technology Engines

9.1 CDS TE API

This section describes the application programming interface provided by the CDS TE. Table 10 describes the set of possible operations. Thanks to these primitives, the CDS TE allows its users to:

- Load/unload ontologies
- Load/unload dictionaries
- Request ontology entities
- Expand metadata
- Get semantically equivalent metadata in a namespace
- Get semantically equivalent query in a namespace
- Fetch ontology
- Fetch dictionary

***Table 10** - CDS TE API*

Service Operation	Origin	Output	Description
loadOntology(ontologyURL)	CoMid client (e.g an Application upon the action of an User or event or some CoMid engine)	success/failure	This operation is called to load an ontology to the CDS. The objectives of this method are • to fetch the ontology • to validate the ontology • to parse the ontology • to store the ontology • to materialize the knowledge of the ontology • check knowledge base consistency
unloadOntology(ontologyURI)	CoMid client (e.g an Application upon the action of an User or event or some CoMid engine)	success/failure	This operation is called to unload an ontology from the CDS. The objectives of this method are • to remove any knowledge on entities of the ontologyURI namespace
loadDictionary(dictionaryURL)	CoMid client (e.g an Application upon the action of an User or event or some CoMid engine)	success/failure	This operation is called to load an dictionary to the CDS. The objectives of this method are • to fetch the dictionary • to validate the dictionary • to parse the dictionary • to store the dictionary and ontologyURIs which it connects • to materialize the knowledge of the dictionary • check knowledge base consistency
unloadDictionary(ontologyURI_1, ontologyURI_2)	CoMid client (e.g an Application upon the action of an	success/failure	This operation is called to unload an dictionary from the CDS. The objectives of this method are • to remove any knowledge connecting entities of the ontologyURI_1 namespace with entities of the

	User or event or some CoMid engine)		ontologyURI_2 namespace
requestOntologyEntity(label, ontologyEntityType, ontologyURI)	CoMid client (e.g an Application upon the action of an User or event or some CoMid engine)	RequestOntologyEntityResultSet	This operation is called to request ontology entities that are stored in the knowledge base of the CDS. The objectives of this method are : • to query knowledge base for ontology entities of ontologyEntityType with URIs inside the ontologyURI namespace and labels matching the specified label • to get the descriptions of the entities • return the entity URIs with their descriptions
expandMetadata(metadata)	Any Orchestrator Engine	Metadata	This operation is called to expand provided metadata according to the knowledge that CDS has. The objectives of this method are • to validate the metadata • to parse the metadata • to materialize the metadata • to serialize the metadata
getEquivalentMetadata(metadata, ontologyURI)	Any Orchestrator Engine	Metadata	This operation is called to get semantically equivalent metadata in the knowledge model identified from the ontologyURI parameter. The objectives of this method are • to validate the metadata • to parse the metadata • to consult the knowledge base to provide semantically equivalent statements in the knowledge model identified by the ontologyURI namespace • to serialize the metadata
getEquivalentQuery(query, ontologyURI)	Any Orchestrator Engine	Query	This operation is called to get semantically equivalent query in the knowledge model identified from the ontologyURI parameter. The objectives of this method are • to validate the query • to parse the query • to consult the knowledge base to provide semantically equivalent criteria in the knowledge model identified by the ontologyURI namespace • to serialize the query
getOntology(ontologyURI)	CoMid client (e.g an Application upon the action of an User or event or some CoMid engine)	Ontology	This operation is called to fetch the ontology stored in the CDS. The objectives of this method is : • return the ontology identified by the ontologyURI
getDictionary(ontologyURI_1, ontologyURI_2)	CoMid client (e.g an Application upon the action of an User or event or some CoMid engine)	Dictionary	This operation is called to fetch the dictionary stored in the CDS. The objectives of this method is : • return the dictionary that connects entities of the ontologyURI_1 namespace with entities of the ontologyURI_2 namespace

9.2 *CoNet TE API*

The CoNet TE API is described by a Java interface, whose methods are listed in the table below.

Table 11 – CoNet TE API

Service Operation	Origin	Output	Description
storeAndAdvertise (NID nid, int rType, String fileName)	Any CoMid Engine	Success/failure	It stores a resource in the CoNet with a given NID, assuming that the localnode is a Conet node that acts as a permanent storage of the resource. It also requests the CoNet to perform CoNet advertising in order to associate the NID with the resource.
storeAndAdvertise (NID nid, int rType, String fileName, NID hostingNid)	Any CoMid Engine	Success/failure	It stores a resource in the CoNet with a given NID, providing the hosting NID to indicate where the resource has to be stored. It also requests the CoNet to perform CoNet advertising in order to associate the NID with the resource.
revoke (NID nid, int rType) ;	Any CoMid Engine	Success/failure	It revokes a content from the CoNet with a given NID, assuming that the local node is the CoNet node that acts as a storage of the resource.
revoke (NID nid, int rType, NID hostingNid) ;	Any CoMid Engine	Success/failure	It revokes a content from the CoNet with a given NID, providing the hosting NID to indicate where the resource was stored.
get (NID nid, String localFileName) throws ConetIOException;	Any CoMid Engine	Success/failure	It retrieves a resource from the CoNet corresponding to the input NID. It stores the named-resource in a local file. It is blocking, i.e. the method call does not exist until the resource is retrieved or an exception is thrown.
advertiseSap (NID nid, SAPListener callBack);	Any CoMid Engine	Success/failure	It advertises a SAP for receiving unNamed-data, providing the callBack to be invoked when unNamed-data are received. By default, the received data will be delivered as byte array.
advertiseSap (NID nid, SAPListener callBack, int localDeliveryType) ;	Any CoMid Engine	success/failure	It advertises a SAP for receiving unNamed-data, providing the callBack to be invoked when unNamed-data are received. The localDelivery type parameter allows to specify how the received unNamed-data can be handled (e.g. as a byte array or a string).
sendToName (NID nid, byte [] unNamedData) ;	Any CoMid Engine	success/failure	It sends unNamed-data towards a NID. The unNamed-data are expressed as a byte array.
sendToName (NID nid, String unNamedDataAsString) ;	Any CoMid Engine	success/failure	It sends unNamed-data towards a NID. The uNnamed-data are expressed as a String, this means that the receiving entity should be ready to receive the data as String.

sendToLocation (LID lid, byte [] unNamedData) ;	Any CoMid Engine	success/failure	It sends unNamed-data towards a LID. The uNamed-data are expressed as a byte array. NB This method will not be implemented in the first release of the CoNet.
sendToLocation (LID lid, String unNamedDataAsString) ;	Any CoMid Engine	success/failure	It sends unNamed-data towards a LID<BR> the unNamed-data are expressed as a String this means that the receiving entity is ready to receive the data as String. NB This method will not be implemented in the first release of the CoNet.

9.3 *Digital Item TE*

This section describes the Application Programming Interface provided by the Digital Item TE. Table 12 describes the set of possible operations. Thanks to these primitives, the Digital Item TE allows its users to:

- Create a fully formed signed DID;
- Create a didl:Item;
- Create a didl:Descriptor;
- Create a didl:Component:
- Create a didl:Resource:
- Add a base didl:Item to a DID;
- Remove a base didl:Item from a DID;
- Add a base didl:Descriptor to a DID;
- Remove a base didl:Descriptor from a DID;
- Add a child didl:Item to a parent didl:Item;
- Remove a child didl:Item from a parent didl:Item;
- Add a didl:Descriptor to a didl:Item;
- Remove a didl:Descriptor from a didl:Item;
- Add a didl:Component to a didl:Item;
- Remove a didl:Component from a didl:Item;
- Add a didl:Descriptor to a didl:Component;
- Remove a didl:Descriptor from adidl:Component;
- Add a didl:Resource to a didl:Component;
- Remove a didl:Resource from adidl:Component;
- Retrieve a specific didl:Item from aDID or fragment of it;
- Retrieve a specific didl:Descriptor from aDID or fragment of it;
- Retrieve a specific didl:Component from aDID or fragment of it;
- Retrieve a specific didl:Resource from aDID or fragment of it;

Table 12 – Digital Item TE API

Service Operation	Origin	Output	Description
createDID(descriptors,items,id,signatureData,declFilePath)	CoMid client (an Application upon the action of a User)	The built didl:DIDL object which is the root of the DID.	This operation is used to create a DI Declaration carrying object. The built object is serialized, signed and written to disk. ***descriptors*** is a collection of didl:Descriptor objects that affect information to the overall DID. ***items*** is a collection of all the didl:Item objects to be stored in the DID. ***id*** is the desired identifier of the DID. ***signatureData*** is an object carrying the necessary information for the production of the DIDs signature (i.e. hash calculation algorithm, ciphering algorithm, ciphering key). ***declFilePath*** is the location, within the file system, where the DID will be written.
createItem(descriptors,items,id)	CoMid client (an Application upon the action of a User)	The built didl:Item object.	This operation is used to create a didl:Item carrying object. ***descriptors*** is a collection of didl:Descriptor objects that affect information to didl:Item to be built. ***items*** is the collection of all the didl:Item objects to be contained within the didl:Item to be created. ***id*** is the desired identifier of the didl:Item.
createDescriptor(content,id)	CoMid client (an Application upon the action of a User)	The built didl:Item object.	This operation is used to create a didl:Descriptor carrying object. ***content*** is the text data to be contained in the didl:Descriptor to be built. ***id*** is the desired identifier of the didl:Descriptor.
createComponent(resource,id)	CoMid client (an Application upon the action of a User)	The built didl:Component object.	This operation is used to create a didl:Component carrying object. ***resource*** is a didl:Resource carrying object that will become a child of the didl:Component to be built. ***id*** is the desired identifier of the didl:Component.
createResource(reference)	CoMid client (an Applicatio	The built didl:Resource object.	This operation is used to create a didl:Resource carrying object. ***reference*** is the value of the reference

	n upon the action of a User)		that points to the digital resource that is represented by the built didl:Resource.
addItem(itemParent,item)	CoMid client (an Application upon the action of a User)	success/failure	This operation adds a didl:Item to the didl element (object) specified as its parent. ***itemParent*** is the object that is to have added to it, as a child, a the given didl:Item. It may be a didl:Container or a didl:Item. ***item*** the didl:Item to be added.
removeItem(itemParent,itemID)	CoMid client (an Application upon the action of a User)	success/failure	This operation removes a didl:Item child from the a specified didl element (object). ***itemParent*** is the object that is to have removed from it a didl:Item child. It may be a didl:Container or a didl:Item. ***itemID*** the identifier of the didl:Item to be removed.
addDescriptor(descriptorParent,descriptor)	CoMid client (an Application upon the action of a User)	success/failure	This operation adds a didl:Descriptor to the didl element (object) specified as its parent. ***descriptorParent*** is the object that is to have added to it, as a child, a the given didl:Descriptor. It may be a didl:Container, a didl:Item or a didl:Component. ***descriptor*** the didl:Descriptor to be added.
removeDescriptor(descriptorParent, descriptorID)	CoMid client (an Application upon the action of a User)	success/failure	This operation removes a didl:Descriptor child from the a specified didl element (object). ***descriptorParent*** is the object that is to have removed from it a didl:Descriptor child. It may be a didl:Container, a didl:Item or a didl:Component. ***descriptorID*** the identifier of the didl:Descriptor to be removed.
addComponent(componentParent,component)	CoMid client (an Application upon the action of a User)	success/failure	This operation adds a didl:Component to the didl element (object) specified as its parent. ***componentParent*** is the object that is to have added to it, as a child, a the given didl:Component. It must be a didl:Item. ***component*** the didl:Component to be added.

removeComponent(componentParent, componentID)	CoMid client (an Application upon the action of a User)	success/failure	This operation removes a didl:Component child from the a specified didl element (object). ***componentParent*** is the object that is to have removed from it a didl:Component child. It must be a didl:Item. ***componentID*** the identifier of the didl:Component to be removed.
addResource(resourceParent, resource)	CoMid client (an Application upon the action of a User)	success/failure	This operation adds a didl:Resource to the didl element (object) specified as its parent. ***resourceParent*** is the object that is to have added to it, as a child, a the given didl:Resource. It must be a didl:Component. ***resource*** the didl:Resource to be added.
removeResource(resourceParent, resourceID)	CoMid client (an Application upon the action of a User)	success/failure	This operation removes a didl:Resource child from the a specified didl element (object). ***resourceParent*** is the object that is to have removed from it a didl:Resource child. It must be a didl:Component. ***resourceID*** the identifier of the didl:Resource to be removed.
getDIDElement(didObject,elementID)	CoMid client (an Application upon the action of a User)	The specified didl object	This operation retrieves the specified didl element from the given did or did sub-element. ***didObject*** is the did information container from which the specified didl element will be retrieved. It may be a didl:DIDL, didl:Container, didl:Item or a didl:Component. ***elementID*** the identifier of the desired didl element.

9.4 *Event Report TE API*

This section describes the Application-programming interface provided by the Event Report TE. Table 13 describes the set of possible operations. Thanks to these primitives, the Event Report TE allows its users to:

- Create ER-Rs;
- Parse and Retrieve data from an ER-R;
- Create Ers;
- Notify users with Ers;
- Parse and Retrieve data from an ER.

Table 13 - Event Report TE API

Service Operation	Origin	Output	Description
CreateERR (ID_ERR, ID_VDI, ERR)	CoMid client (e.g an Application upon the action of an User)	success/failure	This operation is used to create an Event Report Request for a VDI. ID_ERR is the identifier which uniquely identifies this ER-R; ID_VDI is the identifier of the VDI for which the ER-R is created; ERR is the Event Report Request, which will comprise, at least: • A description of the Event; • The syntax/format of the Event Report; • The recipient(s) of the Event Report; • Parameters related to delivery of the Event Report (e.g. transport mechanism, encryption, acknowledgements, etc.). After the Creation of the new ER-R, Event Reports are eligible to be generated upon the occurrence of events described in the ER-R.
Get ERR (ID_ERR)	CoMid client (e.g an Application upon the action of an User)	ERR	This operation is used to retrieved the ERR message using his unique identifier.
CreateER (ID_ER, ID_ERR, ER)	CoMid (e.g the Middleware upon the occurrence of an Event)	success/failure	This operation is used to create an Event Report based on a ER-R. ID_ER is the identifier which uniquely identifies this ER; ID_ERR is the identifier of the ER-Rupon which the ER is created; ER is the Event Report which will comprise of, at least: • A description of the format for delivery and of the access rights to the data of the Report; • The data of the Event Report;
Notify User (ID_ER, ER, recipient)	CoMid	success/failure	This operation is used to send to an User an Event Report which was generated upon the occurrence of an Event (using the description from the Event Report Request). ID_ER is the identifier which uniquely identifies this ER; ER is the generated Event Report which will comprise of the data of the ER (see Table 2); Recipient is the address of the User
Get ER (ID_ER)	CoMid client (e.g an Application upon the action of an User)	ER	This operation is used to retrieved the ER message using his unique identifier.

9.5 Media Framework TE API

The Media Framework TE API is described by a Java interface, whose methods are listed in the table below.

Table 14– Media Framework TE API

Service Operation	Origin	Output	Description
createResource (tools, srcLocation, destLocation)	Any Orchestrator Engine	success/failure	This operation is used to create and handle a given resource. • tools: one or more tools (i.e. IPMP Tool) used to manipulate the resource (e.g. ciphering, filtering). They include description of: o control points involved o name or ID of the tool o one or a set of keys • srcLocation: where the resource handled is located • destLocation: where the resource handled will be located
consumeResource(tools, inputStreamResource, mediaType, renderingObject);	Any Orchestrator Engine	success/failure	This method is used to consume a given resource. • tools: one or more tools (i.e. IPMP Tool) used to manipulate the resource (e.g. ciphering, filtering). They include description of: o control points involved o name or ID of the tool o one or a set of keys • inputStreamResource: the inputStream of the resource • mediaType: what kind of media is handled • renderingObject: where the resource will be rendered (e.g. handler of a rendering window)
setRenderingWindow(renderingObject)	Any Orchestrator Engine	success/failure	This method is used to set a rendering window asynchronously • renderingObject: where the resource will be rendered (e.g. an handler of a rendering window)
performUI (userInteraction)	Any Orchestrator Engine	success/failure	This method is used to request an action on the resource (e.g. play, pause, stop, mute…) • userInteraction: the kind of operation to perform

9.6 Match TE API

This section describes the application programming interface provided by the Match TE. Table 15 describes the set of possible operations. Thanks to these primitives, the Match TE allows its users to:

- Match publications with subscriptions
- Match subscriptions with publications
- Revoke publications from publications table
- Revoke subscriptions from subscriptions table.

Table 15 - Match TE API

Service Operation	Origin	Output	Description
matchPubsWith(S_VDI_ID, subscriptionQuery, expirationDate)	Any Orchestrator Engine	List<P-VDI-Id>	This operation is called whenever a new subscription (S-VDI) arrives to a peer. The objectives of this method are • to store the provided subscription query in the subscription tables, which is identified by the S_VDI_ID identifier, and record the expiration date of the subscription • perform the subscription query against the publication tables • return the publication ids (P-VDI-Id) that where matched.
mathcSubsWith(P_VDI_ID, publicationNetadata, expirationDate)	Any Orchestrator Engine	List<S-VDI-Id>	This operation is called whenever a new publication (P-VDI) arrives to a peer. The objectives of this method are • to store the provided publicationMetadata in the publication tables, which are identified by the P_VDI_ID identifier, and record the expirationDate of the publication • perform subscription Queries from the subscription tables against the newly received publication • return the subscription ids (S-VDI-Id) that where matched.
revokeSubscription(S_VDI_ID)	Any Orchestrator Engine	success/failure	This operation is called whenever a subscription revocation procedure is performed. The objective of this method is • to remove any data recorded about the subscription identified with S_VDI_ID
revokePublication(P_VDI_ID)	Any Orchestrator Engine	success/failure	This operation is called whenever a publication revocation procedure is performed. The objective of this method is • to remove any data recorded about the publication identified with P_VDI_ID

9.7 *Metadata TE API*

This section describes the Application Programming Interface provided by the Metadata TE. Table 16 describes the set of possible operations. Thanks to these primitives, the Metadata TE allows its users to:

- Create:
  - mpeg7:CreationCoordinates elements;
  - mpeg7:CreationDescription elements:
  - mpeg7:Creator elements;
  - mpeg7:Genre elements;

- mpeg7:ParentalGuidanceelements;
- mpeg7:TitleMedia elements.

- Parse:
    - mpeg7:CreationCoordinates elements;
    - mpeg7:CreationDescriptionelements;
    - mpeg7:Creator elements;
    - mpeg7:Genre elements;
    - mpeg7:ParentalGuidance elements;
    - mpeg7:TitleMedia elements.

Table 16 – Metadata TE API

Service Operation	Origin	Output	Description
generateCreationCoordinatesElem (location, date)	CoMid client (an Application upon the action of a User)	The mpeg7:CreationCoordinatesElement/ null	This operation is used to create an mpeg7:CreationCoordinatesElement. location is data structure containing all the necessary data to describe a specific location. The internals of said description are in accordance with the type mpeg7:PlaceType. date is data structure containing all the necessary data to describe a specific time instant. The internals of said data structure are in accordance with the type mpeg7:TimeType.
generateCreationDecriptionElem (creation,classification,relatedMaterial)	CoMid client (an Application upon the action of a User)	The mpeg7:CreationDescriptionElement/ null	This operation is used to create an mpeg7:CreationDescriptionElement. creation is data structure containing all the necessary data to describe a specific media object creation event. The internals of said description are in accordance with the type mpeg7:CreationType. classification is data structure containing all the necessary data to classify a specific media object (genre, language, etc). The internals of said data structure are in accordance with the type mpeg7:ClassificationType. relatedMaterial is data structure containing information related to the mpeg7:CreationDescription element to be generated. The internals of said data structure are in accordance with the type mpeg7:RelatedMaterialType.
generateCreatorElem (character, instrument)	CoMid client (an Application upon the action of a User)	The mpeg7:CreatorElement/null	This operation is used to create an mpeg7:CreatorElement. character is data structure containing all the necessary information to identify a human person. The internals of said data structure are in accordance with the type mpeg7:PersonNameType. instrument is data structure containing all the necessary information to describe a media

			object production tool. The internals of said data structure are in accordance with the type mpeg7:CreationToolType.
generateGenreElem (genre,isPrimaryGenre)	CoMid client (an Application upon the action of a User)	The mpeg7:GenreElement/null	This operation is used to create an mpeg7:GenreElement. genre is the genre's designation. isPrimaryGenre is a boolean value, which, if true, signals that this mpeg7:Genre element represents the primary genre of some media object.
generateParentalGuidanceElem (parentalRating,minimumAge,region)	CoMid client (an Application upon the action of a User)	The mpeg7:ParentalGuidanceElement/null	This operation is used to create an mpeg7:ParentalGuidanceElement. parentalRating is data structure containing all the necessary information to qualify the allowed audience type rating of a media object. The internals of said data structure are in accordance with the type mpeg7:ControlledTermUseType. minimumAge is a non-negative integer. region is a string defining the region of validity of the mpeg7:ParentalGuidance element to be created. It is defined in accordance with type mpeg7:regionCode.
generateTitleMediaElem (titleImage,titleVideo,titleAudio)	CoMid client (an Application upon the action of a User)	The mpeg7:TitleMediaElement/null	This operation is used to create an mpeg7:TitleMediaElement. titleImage is data structure containing all the necessary informationto locate the image which serves as title for some media object. The internals of said data structure are in accordance with the type mpeg7:ImageLocatorType. titleVideo is data structure containing all the necessary informationto locate the video segment which serves as title for some media object. The internals of said data structure are in accordance with the type mpeg7:TemporalSegmentLocatorType. titleAudio is data structure containing all the necessary informationto locate the audio segment which serves as title for some media object. The internals of said data structure are in accordance with the type mpeg7:TemporalSegmentLocatorType.
getCreationCoordinatesData()	CoMid client (an Application upon the action of a User)	The data structures location and date.	This operation is used to retrieve the internal data of an mpeg7:CreationCoordinatesElement. location is data structure containing all the necessary data to describe a specific location. The internals of said description are in accordance with the type mpeg7:PlaceType. date is data structure containing all the necessary data to describe a specific time instant. The internals of said data structure are in accordance with the type mpeg7:TimeType.
getCreationDecriptionData()	CoMid client (an Application upon the action of a	The data structures creation, classification and	This operation is used to retrieve the internal data of an mpeg7:CreationDescriptionElement. creation is data structure containing all the necessary data to describe a specific media

	User)	relatedMaterial.	object creation event. The internals of said description are in accordance with the type mpeg7:CreationType. classification is data structure containing all the necessary data to classify a specific media object (genre, language, etc). The internals of said data structure are in accordance with the type mpeg7:ClassificationType. relatedMaterial is data structure containing information related to the mpeg7:CreationDescription element to be generated. The internals of said data structure are in accordance with the type mpeg7:RelatedMaterialType.
getCreatorData()	CoMid client (an Application upon the action of a User)	The data structures character and instrument.	This operation is used to retrieve the internal data of an mpeg7:CreatorElement. character is data structure containing all the necessary information to identify a human person. The internals of said data structure are in accordance with the type mpeg7:PersonNameType. instrument is data structure containing all the necessary information to describe a media object production tool. The internals of said data structure are in accordance with the type mpeg7:CreationToolType.
getGenreData ()	CoMid client (an Application upon the action of a User)	The data structures genre and isPrimaryGenre.	This operation is used to retrieve the internal data of an mpeg7:GenreElement. genre is the genre's designation. isPrimaryGenre is a boolean value, which, if true, signals that this mpeg7:Genre element represents the primary genre of some media object.
getParentalGuidanceData ()	CoMid client (an Application upon the action of a User)	The data structures parentalRating, minimumAge and region.	This operation is used to retrieve the internal data of an mpeg7:ParentalGuidanceElement. parentalRating is data structure containing all the necessary information to qualify the allowed audience type rating of a media object. The internals of said data structure are in accordance with the type mpeg7:ControlledTermUseType. minimumAge is a non-negative integer. region is a string defining the region of validity of the mpeg7:ParentalGuidance element to be created. It is defined in accordance with type mpeg7:regionCode.
getTitleMediaData()	CoMid client (an Application upon the action of a User)	The data structures titleImage, titleVideo and titleAudio.	This operation is used to retrieve the internal data of anmpeg7:TitleMediaElement. titleImage is data structure containing all the necessary informationto locate the image which serves as title for some media object. The internals of said data structure are in accordance with the type mpeg7:ImageLocatorType. titleVideo is data structure containing all the necessary informationto locate the video segment which serves as title for some media object. The internals of said data structure are in accordance with the type mpeg7:TemporalSegmentLocatorType.

			titleAudio is data structure containing all the necessary informationto locate the audio segment which serves as title for some media object. The internals of said data structure are in accordance with the type mpeg7:TemporalSegmentLocatorType.

9.8 MPEG-21 File TE API

This section describes the Application Programming Interface provided by the MPEG-21 File TE. Table 17 describes the set of possible operations. Thanks to these primitives, the MPEG-21 File TE allows its users to:

- Create MPEG-21 files/archives;
- Add metadata files to already existing MPEG-21 files/archives;
- Add resource (media) files to already existing MPEG-21 files/archives;
- Replace files within already existing MPEG-21 files/archives;
- Remove files from within already existing MPEG-21 files/archives;
- Obtain a manifest of all the internal contents of an MPEG-21 file/archive.
- Obtain a specific inner component of an MPEG-21 file/archive.

Table 17 – MPEG-21 File TE API

Service Operation	Origin	Output	Description
createMPEG21File(mpeg21FileName, didFileLocalPath, metadataFilesLocalPaths, resourceFilesLocalPaths)	CoMid client (an Application upon the action of a User)	local path of the produced MPEG-21 file/null	This operation is used to create an MPEG-21 file which is stored on the local disk. mpeg21FileName is the desired name for the MPEG-21 file to be produced didFileLocalPaththe local path of the DID file to be added to the MPEG-21 file to be produced; metadataFilesLocalPaths is a collection object containing the local paths of all the metadata files to be included in the MPEG-21 file; resourceFilesLocalPaths – is a collection object containing the local paths of all the resource (media) files to be included in the MPEG-21 file;
addMetadataToMPEG21File(mpeg21FileLocalPath, newMetadataFilesLocalPaths)	CoMid client (an Application upon the action of a User)	local path of new version of the MPEG-21 file/null	This operation is used to add one or more metadata files to an already existing MPEG-21 file. The old version (on the local disk), of the MPEG-21 file will be replaced with its new version, containing the new file(s); mpeg21FileLocalPath is the local path of the MPEG-21 file/archive to which the new metadata files are to be added newMetadataFilesLocalPathsis a collection object containing the local

			paths of the metadata files that are to be added to the MPEG-21 file/archive;
addResourceToMPEG21File (mpeg21FileLocalPath, newResourceFilesLocalPaths)	CoMid client (an Application upon the action of a User)	local path of new version of the MPEG-21 file/null	This operation is used to add one or more resource files to an already existing MPEG-21 file. The old version (on the local disk), of the MPEG-21 file will be replaced with its new version, containing the new file(s); mpeg21FileLocalPath is the local path of the MPEG-21 file/archive to which the new resource files are to be added; newResourceFilesLocalPaths is a collection object containing the local paths of the files that are to be added to the MPEG-21 file/archive;
replaceMPEG21FileComponent(mpeg21FileLocalPath, replacingComponentFileLocalPath, replacedComponentFileName)	CoMid client (an Application upon the action of a User)	local path of new version of the MPEG-21 file/null	This operation is used to replace an internal component (file) of an already existing MPEG-21 file. The old version (on the local disk), of the MPEG-21 file will be replaced with its new version, containing the new file. The added component file's compression status will be the same as that of the replaced file. mpeg21FileLocalPath is the local path of the MPEG-21 file/archive which is going to have one of its components altered; replacingComponentFileLocalPath is the local path of the replacing component file; replacedComponentFileName is the name of the file (contained inside the MPEG-21 file), that is to be replaced;
removeFromMPEG21File(mpeg21FileLocalPath, removedFileName)	CoMid client (an Application upon the action of a User)	local path of new version of the MPEG-21 file/null	This operation is used to remove an internal component (file) of an already existing MPEG-21 file. The old version (on the local disk), of the MPEG-21 file will be replaced with its new version. mpeg21FileLocalPath is the local path of the MPEG-21 file/archive from which a component file will be removed; removedFileName is the name of the file that is to be removed from the MPEG-21 file/archive;
getMPEG21FileContentManifest(mpeg21FileLocalPath)	CoMid client (an Application upon the action of a User)	file manifest/null	This operation is used to obtain a list of all of an MPEG-21 archive's inner files; mpeg21FileLocalPath is the local path of the MPEG-21 file/archive from which a manifest is to be obtained;
getMPEG21FileComponent (mpeg21FileLocalPath, mpeg21FileComponentID)	CoMid client (an Application upon the action of a User)	the specified inner component (file) of an MPEG-21 file7null	This operation is used to obtain an object representing the specified MPEG-21 file's inner file- mpeg21FileLocalPath (string) – the local path of the MPEG-21 file/archive from which a component file will be retrieved; mpeg21FileComponentID (string) – the identifier of the desired inner component of an MPEG-21 file;;

9.9 Overlay TE API

This section describes the Application Programming Interface provided by the Overlay TE.

9.9.1 MessageCreator

Operation	Origin	Output	Description
createMessage(header, payload)	CoMid client (e.g an Application upon the action of an User or event or some CoMid engine)	Overlay message	This operation creates a message to be circulated in the Overlay, formatted according to Figure 29.
addHeader(message, header)	CoMid client (e.g an Application upon the action of an User or event or some CoMid engine)	Overlay message	This operation adds the header part of the message.
addPayload(message, payload)	CoMid client (e.g an Application upon the action of an User or event or some CoMid engine)	Overlay message	This operation adds the payload part of the message.

9.9.2 MessageKeeper

Operation	Origin	Output	Description
bufferMessage(overlayMessage)	CoMid (e.g some CoMid engine)	Success/failure	This operation stores the received overlay message in the peer's buffer.
getBufferedMessages()	CoMid (e.g some CoMid engine)	List of all the buffered messages	This operation returns a list with all the messages contained in this buffer.
unbufferMessage(message)	CoMid (e.g some CoMid engine)	Success/failure	This operation removes all messages matching the given parts of this message (if a part is missing, then it is not checked) from the buffer.

9.9.3 MessageParser

Operation	Origin	Output	Description
getMessageHeader(message)	CoMid client (e.g an Application upon the action of an User or event or some CoMid engine)	Message header	This operation returns the header part of the message.
getMessagePayload(message)	CoMid client (e.g an Application upon the action of an User or event or some CoMid engine)	Message payload	This operation returns the payload part of the message.

9.9.4 PropagationMessageHandler

Operation	Origin	Output	Description
sendPropagationMessage(overlay	CoMid (e.g some CoMid	Success/failure	This operation sends an overlay to a remote peer. The sender parses the payload to check whether this is a

Message)	engine)		publication or a subscription VDI and, then, extracts the corresponding service endpoint of the remote peer from the registry.
recvPropagationMessageDaemon()	Overlay	Success/failure	This operation runs the services daemons for receiving publication and subscription VDIs. Upon receipt, the messages are consumed.
consumePropagationMessage(overlayMessage)	Overlay	Success/failure	This operation parses the received message, checks whether this is a publication or a subscription VDI and, then, stores it in the corresponding buffer.

9.9.5 RegistryCreator

Operation	Origin	Output	Description
registerToFractal(fractalId)	CoMid client (e.g an Application upon the action of an User or event or some CoMid engine)	Success/failure	This operation initiates the registration procedure of a peer to a fractal.
notifyNeighbors()	CoMid client (e.g an Application upon the action of an User or event or some CoMid engine)	Success/failure	This operation notifies the other fractal members of this peer's presence.
advertiseRegistry()	CoMid client (e.g an Application upon the action of an User or event or some CoMid engine)	Success/failure	This operation advertises the registry held by this peer to the network.
updateRegistry()	CoMid client (e.g an Application upon the action of an User or event or some CoMid engine)	Registry	This operation retrieves the currently advertised registry from the network and updates the peer's one.

9.9.6 RegistryParser

Operation	Origin	Output	Description
parseFractalRegistry(registry)	CoMid client (e.g an Application upon the action of an User or event or some CoMid engine)	Success/failure	This operation parses the fractal registry and creates internal structures to handle it.
getPublicationServiceEndpoints()	CoMid client (e.g an Application upon the action of an User or event or some CoMid engine)	List of publication service endpoints	This operation reads the registry and returns a list with all the publication service endpoints.
getPublicationServiceEndpoints()	CoMid client (e.g an Application upon the action of an User or event or some CoMid engine)	List of subscription service endpoints	This operation reads the registry and returns a list with all the subscription service endpoints.
getPublicationServiceEndpoints(peerId)	CoMid client (e.g an Application upon the action of an User or event or some CoMid engine)	The publication service endpoint of the given peer.	This operation reads the registry and returns the publication service endpoint of the given peer.
getSubscriptionServiceEndpoints(	CoMid client (e.g an Application upon the	The subscription	This operation reads the registry and returns the subscription service endpoint of the given peer.

peerId)	action of an User or event or some CoMid engine)	service endpoint of the given peer.	
insertPeerToRegistry(peerId, publicationServiceEndpoint, ubscriptionServiceEndpoint, leaveDate)	CoMid client (e.g an Application upon the action of an User or event or some CoMid engine)	Success/failure	This operation inserts a peer into the local registry.
removePeerFromRegistry(peerId, publicationServiceEndpoint, ubscriptionServiceEndpoint)	CoMid client (e.g an Application upon the action of an User or event or some CoMid engine)	Success/failure	This operation removes a peer from the local registry.

9.10 REL TE API

This section describes the application programming interface provided by the REL TE. The sections below describes the set of possible operations.

9.10.1 Rights expression creation

9.10.1.1 LicenseCreator

Operation	Origin	Output	Description
setLicenseID(LicenseID)	Any CoMid engine	Success/failure	Calling this method results in adding an identifier to the license that will be generated.
setEncryptedLicense()	Any CoMid engine	Success/failure	Sets the encrypted license property of a license
addIssuer (Issuer issuer)	Any CoMid engine	Success/failure	Calling this method results in adding anissuer to the license that will be generated.
addGrants(List<Grant> grants)	Any CoMid engine	Success/failure	Calling this method results in adding a Grant to the license that will be generated.
setLicenseSignature(Signature)	Any CoMid engine	Success/failure	Calling this method results in settingthesignature of the license that will be generated
setInvetory(Inventory inventory)	Any CoMid engine	Success/failure	Calling this method results in settingthe inventory to the license that will be generated.
createFile(String Filepath)	Any CoMid engine	License file	Creates a License file
generateString()	Any CoMid engine	xml representation of a license	Gives the xml representation of a license

9.10.1.2 IssuerCreator

Operation	Output	Description
setPrincipal(Principal principal)	Success/failure	Sets a principal

setSignature(Signature signature)	Success/failure	Sets signature
setDetails(IssuerDetails issuerDetails)	Success/failure	Returns issuer details, like time of issue

9.10.1.3 IdentityHolderCreator

Operation	Output	Description
setIDValue(String idValue)	Success/failure	Sets an ID value for the identiy holder
setIDSystem(String idSystem)	Success/failure	Sets an ID value for the system issuing the license

9.10.1.4 GrantCreator

Operation	Output	Description
setRight (Right right)	Success/failure	Sets the right of the grant
setPrincipal(Principal principal)	Success/failure	Sets the principal of the grant
setConditions(List<Condition>)	Success/failure	Sets the conditions that should be met in order for the grant to be valid
setResource(Resource resource)	Success/failure	Sets the resource of the grant
setDIreference(DIreference diReference)	Success/failure	Sets the reference of a grant to a diital item
setEncryptedContent(EncryptedContent content)	Success/failure	Sets encryption on a grant

9.10.1.5 DigitalResourceCreator

Operation	Output	Description
setNonSecureReferenceURI(String URI)	Success/failure	Sets the non secure reference URI to access the digital resource.
setSecureIndirect(String URI)	Success/failure	Sets a secure reference URI to access the digital resource.

9.10.1.6 KeyHolderCreator

Operation	Output	Description
setKeyInfo(KeyInfo keyInfo)	Success/failure	Sets the info of a key.
setKeyHolderType(KeyHolderType keyHolderType)	Success/failure	Sets the type of a keyholder.

9.10.1.7 ProtectedResourceCreator

Operation	Output	Description
setEncryptedKey(MXMObject encryptedKeyObj)	Success/failure	Sets an encrypted key
setEncryptedData(MXMObject encryptedData)	Success/failure	Sets an encrypted data

9.10.2 Rights expression parsing

9.10.2.1 LicenseParser

Operation	Origin	Output	Description
getLicenseID()	Any CoMid engine	String	Returns the license ID
getInventory()	Any CoMid engine	Inventory	Returns the inventory of a license
getIssuer()	Any CoMid	Issuer	Returns the issuer of a license

	engine		
getEncryptedLicense()	Any CoMid engine	Encrypted License	Returns the encrypted content of a license
getLicenseSignature()	Any CoMid engine	Signature	Returns the signature of a license
getGrants()	Any CoMid engine	List of Grants	Returns the list of Grants expressed in the license
parse(FilePath)	Any CoMid engine	License object	Parses a license

9.10.2.2 IssuerParser

Operation	Origin	Output	Description
getPrincipal()	Any CoMid engine	Principal	Returns the Principal
getSignature()	Any CoMid engine	Signature	Returns the Signature
getDetails()	Any CoMid engine	Issuer details	Returns issuer details, like time of issue.

9.10.2.3 IdentityHolderParser

Operation	Origin	Output	Description
getIDValue()	Any CoMid engine	String	Returns the ID value of the identity holder
getIDSystem()	Any CoMid engine	String	Returns the ID value of the system that issued the license

9.10.2.4 GrantParser

Operation	Origin	Output	Description
getPrincipal()	Any CoMid engine	Principal	Returns the Principal of the Grant
getResource()	Any CoMid engine	Resource object	Returns the resource object of a Grant
getEncryptedContent()	Any CoMid engine	EncryptedContent	Returns the EncryptedContent of a Grant
getConditions()	Any CoMid engine	List<Condition>	Returns the conditions that should be met in order to the grant to be valid
getRight()	Any CoMid engine	Right	Returns the right that the grant grants to the principal
geDIReference()	Any CoMid engine	DIReference	Returns the reference of a grant to a digital item

9.10.2.5 DigitalResourceParser

Operation	Origin	Output	Description
getNonSecureReferenceURI()	Any CoMid engine	URI	Returns the non secure reference URI to access the digital resource that is stated in the license
getSecureIndirect()	Any CoMid engine	URI	Returns the secure reference URI to access the digital resource that is stated in the license

9.10.2.6 KeyHolderParser

Operation		Output	Description
getKeyInfo()	Any CoMid engine	KeyInfo	Returns info of the key.
getKeyHolderType()	Any CoMid engine	KeyholderType	Returns the type of a keyholder

9.10.2.7 ProtectedResourceParser

Operation	Origin	Output	Description
getEncryptedData(key)	Any CoMid engine	EncryptedData	Returns the encrypted data
getEncryptedKey(String keyType)	Any CoMid engine	String	Returns the encrypted key

9.10.3 Authorization Manager

Operation	Origin	Output	Description
verifyGrant(Grant licenseGrant, MXMObject principal,Right right, MXMObject resource)	Any CoMid engine	Success/failure	Verifies whether the Grant in the supplied license grants rights over the resource or the principal specified in the query as well as if the Grant grants the Right right.
authorise(License license, License query)	Any CoMid engine	Success/failure	This method receives a License and a Query and returns the result of the validation
verifyRight (Grant licenseGrant, Right right)	Any CoMid engine	Success/failure	Verifies a specific right over the license Grant
verifyResource(Grant licenseGrant, MXMObject resource)	Any CoMid engine	Success/failure	Verifies a specific resource

9.10.4 Condition Manager

Operation	Origin	Output	Description
verifyConditions (List<MXMObject> conditions, QName currentCountry, QName currentRegion, boolean isCommercial)	Any CoMid engine	Success/failure	Verifies if a list of conditions are met
verifyValidityInterval(ValidityInterval validityInterval)	Any CoMid engine	Success/failure	Verifies if the license period is within the specified interval
verifyLocation (Location licenseLocation, QName currentCountry, QName currentRegion)	Any CoMid engine	Success/failure	Verifies the country or region conditions

9.11 Security TE API

The security TE API is extracted from the MPEG specification available at:

http://mxm.wg11.sc29.org/docs/api/java/d2/da3/namespaceorg_1_1iso_1_1mpeg_1_1mxm_1_1engine_1_1securityengine.html

As clarified in section 7.2.1, the inclusion of this API in the ANNEX is not meant as a commitment to implement it in this exact status. Rather it is meant to document the starting point of the analysis to design a Security TE API tailored to the needs of the CONVERGENCE system. In particular, we have started pointing out some issues that CONVERGENCE should address, as reported in section 9.11.2.

In the package org.iso.mpeg.mxm.engine.securityengine we find the following interfaces and classes:

interface	CertificateManager
interface	KeyManager
interface	SecureDeviceManager
interface	SecureRepositoryManager
class	SecurityEngine
interface	SecurityEngineKeys
interface	SecurityInfoCreator
interface	SecurityInfoParser

9.11.1 CertificateManager Interface

Classes implementing this interface are responsible for providing a number of security-related functionalities such as the generation of private/public keys, import and export of public key and certificates in various formats, of cryptographic services, secure storage and retrieval of information, generation of keys, signature calculation and validation, etc.

Operation	Description
boolean isAlgorithmSupported (String algorithmURI)	This method is used to query whether an object implementing this interface supports a certain algorithm Parameters: algorithmURI the URI identifying a cryptographic algorithm.
MXMObject generateKeyPair (String algorithm) throws CertificateManagerException	Generate a key pair (a private key and associated public key) using the given algorithmParameters: algorithm - algorithm to be used with this key pairReturns: a key pair
void createUserEntry (String keyAlias, String password, MXMObject privateKeyEntry, MXMObject parameters) throws CertificateManagerException	This method is used to add a new user entry in the KeyStore. If the privateKeyEntry parameter is null, a new public/private key pair and a self certificate will be created. Parameters: keyAlias The alias used to Store the new entry in the KeyStore password The password for retrieving the entry in the KeyStore associated to the alias specified privateKeyEntry the PrivateKeyEntry to store in the KeyStore associated to the username. If this parameter is null, a test privateKeyEntry will be generated. parameters A set of parameters which may be used to create the user entry
boolean testCredentials (String keyAlias, String password) throws CertificateManagerException	This method is used to test if the username and password entered by the user is correct. Parameters:

	keyAlias the KeyAlias to be tested password the password to be tested
MXMObject getKeyInfo (String keyAlias, String password) throws CertificateManagerException	This method is used to obtain the certificate associated to a certain keyAlias as an REL Principal Parameters: keyAlias the keyAlias for which the Principal is sought password the password necessary to retrieve the data from the KeyStore
boolean verifyKeyInfo (String keyAlias, String password, MXMObject keyInfo) throws CertificateManagerException, SecurityEngineException	
String importCertificate (String keyAlias, String password, FileInputStream fis) throws CertificateManagerException	This method is employed to import a certificate in the certificate repository managed by the SecurityEngine. Parameters: keyAlias the KeyAlias associated to the certificate which is going to be imported. In case it is null, the CertificateManager will generate one using the certificate data and return it. password the password to access the certificate repository fis the FileInputStream associated to the file where the Certificate to be imported resides Returns: the same keyAlias given in input if not null, or else a keyAlias generated from the certificate data.
MXMObject getCertificate (String keyAlias, String password) throws CertificateManagerException	This method is employed to retrieve the certificate associated to a certain keyAlias from the certificate repository Parameters: alias the alias to which the sought Certificate is bound password the password to access the certificate repository Returns: the certificate corresponding to the keyAlias specified
void deleteCertificate (String keyAlias, String password) throws CertificateManagerException	This method is employed to delete a certificate associated to a certain alias from the certificate repository Parameters: keyAlias the alias associated to the certificate to be deleted password the password to access the certificate repository

9.11.2 KeyManagerInterface

Classes implementing this interface are responsible for providing a number of security-related functionalities such as the generation of symmetric keys, providing generation of keys, hashes, signature calculation and validation, etc.

As regards the integration of this interface in the CONVERGENCE project, we observe an issue with the verify Signature operation. The parameters and description suggest that the verification process for a given digital signature consists of its recreation and subsequent comparison to the given signature. While this might make some sense for "symmetric signatures" (which are rather keyed MACs), it is not suitable for true digital (asymmetric) signatures for a couple of reasons. To begin with, a verifier does not (or better must not) dispose of the private signature key with which the signature has been generated, and therefore cannot recreate it. Secondly, even if he could, two signatures - on exactly the same data, issued under exactly the same algorithm, using exactly the same key - need not be equal (may indeed appear like two totally different values!). The reason for the latter phenomenon lies in the fact that many signature schemes today (unlike RSA PKCS1#1 v1.5) are probabilistic. These considerations will drive the adaptation/extension of this interface towards the needs of CONVERGENCE.

Operation	Description
byte [] org.iso.mpeg.mxm.engine.securityengine.KeyManager.decrypt (byte[] data, String algorithm, String keyAlias, String password) throws KeyManagerException	This method is used to asymmetrically decrypt the data in input using the private key associated to a certain key alias given in input. Parameters: data the data to be decrypted algorithm the algorithm to be used to decrypt the data keyAlias the alias associated to the key to be used decrypt the data in input password the password to access the certificate repository Returns: the decrypted data Exceptions: KeyManagerException
byte [] org.iso.mpeg.mxm.engine.securityengine.KeyManager.encrypt (byte[] data, byte[] encryptionKey, String algorithm) throws KeyManagerException	This method is used to symmetrically encrypt the data in input with a given symmetric key given in input. Parameters: data the data to be encrypted encryptionKey the key to be used to encrypt the data algorithm the algorithm to be used to encrypt the data Returns: the encrypted data as a byte array Exceptions: KeyManagerException
byte [] org.iso.mpeg.mxm.engine.securityengine.KeyManager.encrypt (byte[] data, String algorithm, String keyAlias, String password) throws KeyManagerException	This method is used to asymmetrically encrypt the data in input with a given algorithm using the public key from a certificate stored in the keyStore and identified by a keyAlias. Parameters: data the data to be encrypted algorithm the algorithm to be used to encrypt the data keyAlias an alias associated to the key to be used to encrypt the data password the password for accessing the certificate repository Returns: the encrypted data as a byte array Exceptions: KeyManagerException
byte [] org.iso.mpeg.mxm.engine.securityengine.KeyManager.generateHash (FileInputStream fis, String algorithm) throws KeyManagerException	This method generates a hash value of the data in input using a given algorithm. Parameters: fis the FileInputStream of the file containing the data to be hashed algorithm the algorithm used to calculate a hash value of the data Returns: the hash value Exceptions: KeyManagerException
byte []	This method generates a hash value of the data in input using a

org.iso.mpeg.mxm.engine.securityengine.KeyManager.generateHash (byte[] data, String algorithm) throws KeyManagerException	given algorithm. Parameters: data the data to be hashed algorithm the algorithm used to calculate a hash value of the data Returns: the hash value Exceptions: KeyManagerException
byte [] org.iso.mpeg.mxm.engine.securityengine.KeyManager.generateRandomKey (String algorithm, int keyLength) throws KeyManagerException	Generate a random key using a specific algorithm. Parameters: algorithm the algorithm used to generate the random key keyLength the length of the key to be generated, in bytes. Returns: the generated key Exceptions: KeyManagerException
byte [] org.iso.mpeg.mxm.engine.securityengine.KeyManager.generateSignature (byte[] data, String algorithm, String keyAlias, String password) throws KeyManagerException	This method is used to generate a digital signature for the data in input using the given algorithm and a private key associated to a key alias. Parameters: data the data to be signed algorithm the algorithm used to sign the data keyAlias the alias associated to the private key to be used to sign the data password the password to access the key/certificate repository Returns: the digital signature of the input data Exceptions: KeyManagerException
boolean org.iso.mpeg.mxm.engine.securityengine.KeyManager.isAlgorithmSupported (String algorithmURI)	This method is used to query whether an object implementing this interface supports a certain algorithm. Parameters: algorithmURI the URI identifying a cryptographic algorithm. Returns: true if the algorithm specified is supported
boolean org.iso.mpeg.mxm.engine.securityengine.KeyManager.verifyHash (byte[] data, byte[] hashValue, String algorithm) throws KeyManagerException	This method verifies that the hash value of the data in input has a specific value. Parameters: data the data on which the hash value has to be calculated hashValue the hash value which has to be verified against the one calculated algorithm the algorithm used to calculate the hash value of the data Returns: true if the calculated value is equal to the generated one Exceptions: KeyManagerException
boolean org.iso.mpeg.mxm.engine.securityengine.KeyManager.verifySignature (byte[] data, byte[] signature, String algorithm, String keyAlias, String password) throws KeyManagerException	This method is used to verify that the signature of the data in input is equals to a certain value. Parameters: data the data upon which digital signature has to be calculated signature the value of the signature to be verified algorithm the algorithm used to verify the signature keyAlias the alias associated to the key to be used to generate the signature password the password to access the key/certificate repository Returns: true if the digital signature calculated is equal to that given in

	input Exceptions: KeyManagerException

9.11.3 SecureDeviceManagerInterface

Classes implementing this interface can be used to certify and verify the integrity of MXM Devices, when needed. The normal operation should be as follows:

1. A Device is certified the first time is used. It means, that a fingerprint is calculated with the hardware data and basic software installed. The Device is thus uniquely identified. This information is sent to a secure repository.

2. A Device is verified each time its integrity is critical. Verification recalculates the fingerprint and checks against the Secure repository if the fingerprint has changed.

There are also operations to disable a device and to self-verify.

Operation	Description
MXMObject org.iso.mpeg.mxm.engine.securityengine.SecureDeviceManager.certifydevice (String deviceFingerprint, String userToken)	This method is employed to check the integrity of a software device prior to its first usage. Some features regarding the device (operating system) and/or the software device that is requesting the authorisation of a user action will be registered so that they can be later checked for integrity. Parameters: deviceFingerprint An XML piece that represents the relevant features about the device (operating system) and/or the software device. userToken An XML piece that represents the SAML token, which contains the credentials of the user. Returns: CertificationResult, a class that contains the following information: * Int certificationResult: the result of the certifydevice process * Byte[] devicePKCS12: an X509 certificate and private key that identify the certified device. The X509 certificate includes a unique device identifier and an enabling code that can be used for the selfVerify process.
String org.iso.mpeg.mxm.engine.securityengine.SecureDeviceManager.disableDevice ( )	This method is employed to disable the operation of a software device due to any problem detected during the verifydevice process. The device will be deactivated in the device where it is running. Returns: The result of the disabledevice operation
String org.iso.mpeg.mxm.engine.securityengine.SecureDeviceManager.estimateDeviceFingerprint ( )	This method is employed to extract relevant features about the device (operating system) and/or the software device that is requesting the authorisation of a user action. This information can be used to be registered during the first usage attempt of the device in the system so that it can be later checked for integrity. Returns: an XML piece that represents the relevant features about the device (operating system) and/or the software device

String org.iso.mpeg.mxm.engine.securityengine.SecureDeviceManager.selfVerifyDevice (String deviceFingerprint)	This method is employed to locally check the integrity of a software device during its whole life operation. The features regarding the device are used to compute a security code and determine if it matches the enabling code received during certification. In this way, this method determines whether the device has been manipulated. Parameters: deviceFingerprint An XML piece that represents the relevant features about the device (operating system) and/or the software device. Returns: "OK" if the operation was successful, error message if not.
String org.iso.mpeg.mxm.engine.securityengine.SecureDeviceManager.verifyDevice (String deviceFingerprint, String userToken)	This method is employed to remotely check the integrity of a software device during its whole life operation. Some features regarding the device (operating system) and/or the software device that is requesting the authorisation of a user action are sent to be compared to those registered in the certifydevice process. Parameters: deviceFingerprint An XML piece that represents the relevant features about the device (operating system) and/or the software device. userToken An XML piece that represents the SAML token, which contains the credentials of the user. Returns: The result of the verifydevice process

9.11.4 SecureRepositoryManagerInterface

Classes implementing this interface are used to store, retrieve and manage confidential information in the secure repository.

Operation	Description
void org.iso.mpeg.mxm.engine.securityengine.SecureRepositoryManager.addConfidentialInfo (String alias, MXMObject confidentialInfo, MXMObject secureRepository) throws SecureRepositoryManagerException	This method is used to store generic confidential information into the secure repository. Parameters: alias the alias to be associated to the security information to be stored confidentialInfo the secure information to be stored in the secure repository. Exceptions: SecureRepositoryManagerException
void org.iso.mpeg.mxm.engine.securityengine.SecureRepositoryManager.addLicense (String alias, MXMObject license, MXMObject secureRepository) throws SecureRepositoryManagerException	This method is used to store a license into the secure repository associating an alias to it. Parameters: alias the alias to be associated to the license to be stored license the license to be stored in the secure repository Exceptions: SecureRepositoryManagerException
int org.iso.mpeg.mxm.engine.securityengine.SecureRepositoryManager.deleteConfidentialInfo (String alias, MXMObject secureRepository) throws SecureRepositoryManagerException	This method is used to delete confidential information associated to a certain alias and stored in the secure repository. Parameters: alias the alias associated to the confidential information to be deleted Returns: the number of entries that have been deleted Exceptions:

	SecureRepositoryManagerException
int org.iso.mpeg.mxm.engine.securityengine.SecureRepositoryManager.deleteLicenses (String alias, MXMObject secureRepository) throws SecureRepositoryManagerException	This method is used to delete all licenses associated to a given alias from the secure repository. Parameters: alias the alias associated to the licenses to be deleted Returns: the number of licenses that have been deleted Exceptions: SecureRepositoryManagerException
List<MXMObject> org.iso.mpeg.mxm.engine.securityengine.SecureRepositoryManager.getConfidentialInfo (String alias, String objectType, MXMObject secureRepository) throws SecureRepositoryManagerException	This method is used to retrieve confidential information associated to a certain alias and stored in the secure repository. Parameters: alias the alias associated to the confidential information to be retrieved Returns: all entries of confidential information that have been associated to the given alias Exceptions: SecureRepositoryManagerException
QName org.iso.mpeg.mxm.engine.securityengine.SecureRepositoryManager.getCountry (MXMObject secureRepository) throws SecureRepositoryManagerException	
List<MXMObject> org.iso.mpeg.mxm.engine.securityengine.SecureRepositoryManager.getLicenses (String alias, String ObjectType, MXMObject secureRepository) throws SecureRepositoryManagerException	This method is used to retrieve a list of licenses associated to a certain alias from the secure repository. Parameters: alias the alias associated to one or more licenses stored in the secure repository Returns: all licenses associated to a certain alias Exceptions: SecureRepositoryManagerException
QName org.iso.mpeg.mxm.engine.securityengine.SecureRepositoryManager.getRegion (MXMObject secureRepository) throws SecureRepositoryManagerException	
boolean org.iso.mpeg.mxm.engine.securityengine.SecureRepositoryManager.isCommercial (MXMObject secureRepository) throws SecureRepositoryManagerException	
void org.iso.mpeg.mxm.engine.securityengine.SecureRepositoryManager.setCommercial (MXMObject secureRepository) throws SecureRepositoryManagerException	
void org.iso.mpeg.mxm.engine.securityengine.SecureRepositoryManager.setCountry (QName country, MXMObject secureRepository) throws SecureRepositoryManagerException	
void org.iso.mpeg.mxm.engine.securityengine.SecureRepositoryManager.setRegion (QName region, MXMObject secureRepository) throws SecureRepositoryManagerException	

9.11.5 SecurityEngine Class

Classes implementing this interface are entry point classes for performing security-related functions. From classes implementing the SecurityEngine interface one may get instances of classes performing the main functionalities of this MXM Engine:

- classes to manage digital certificates
- classes to manage symmetric keys and encrypt/decrypt data
- classes to store confidential information in the secure repository
- classes to manage the integrity of MXM tools

Operation	Description
abstract CertificateManager org.iso.mpeg.mxm.engine.securityengine.SecurityEngine.getCertificateManager ( ) [pure virtual]	This method returns an instance of class CertificateManager which is needed to create new credentials and manage certificates. Returns: an object of class CertificateManager or null if this method has not been implemented by the specific SecurityEngine.
abstract KeyManager org.iso.mpeg.mxm.engine.securityengine.SecurityEngine.getKeyManager ( ) [pure virtual]	This method returns an instance of class KeyManager which is needed to generate symmetric keys and to encrypt/decrypt data. Returns: an object of class KeyManager or null if this method has not been implemented by the specific SecurityEngine.
abstract SecureDeviceManager org.iso.mpeg.mxm.engine.securityengine.SecurityEngine.getSecureDeviceManager ( ) [pure virtual]	This method returns an instance of the class SecureToolManager which is needed to certify the integrity of MXM tools. Returns: an object of class SecureToolManager or null if this method has not been implemented by the specific SecurityEngine
abstract SecureRepositoryManager org.iso.mpeg.mxm.engine.securityengine.SecurityEngine.getSecureRepositoryManager ( ) [pure virtual]	This method returns an instance of class SecureRepositoryManager which is needed to store confidential information such as licenses and keys in the secure repository. Returns: an object of class SecureRepositoryManager or null if this method has not been implemented by the specific SecurityEngine.
abstract SecurityInfoCreator org.iso.mpeg.mxm.engine.securityengine.SecurityEngine.getSecurityInfoCreator ( ) [pure virtual]	This method returns an instance of the class SecurityInfoCreator which is used for creating security-related data structures such as those defined by W3C Digital Signature and XML Encryption Returns: an object of class SecurityInfoCreator or null if this method has not been implemented by the specific SecurityEngine
abstract SecurityInfoParser org.iso.mpeg.mxm.engine.securityengine.SecurityEngine.getSecurityInfoParser ( ) [pure virtual]	This method returns an instance of the class SecurityInfoParser which is used for parsing security-related data structures such as those defined by W3C Digital Signature and XML Encryption Returns: an object of class SecurityInfoParser or null if this method has not been implemented by the specific SecurityEngine

9.11.6 SecurityEngineKeys Interface

Operation	Description
final String org.iso.mpeg.mxm.engine.securityengine.SecurityEngineKeys.ASYMMETRIC_ENCRYPTION_ALGORITHM = "org.iso.mpeg.mxm.engine.securityengine.asymmetric.encryption.algorithm" [static]	
final String org.iso.mpeg.mxm.engine.securityengine.SecurityEngineKeys.ASYMMETRIC_ENCRYPTION_KEY_LENGTH = "org.iso.mpeg.mxm.engine.securityengine.asymmetric.encryption.key.length" [static]	
final String org.iso.mpeg.mxm.engine.securityengine.SecurityEngineKeys.CERTIFICATE_DURATION = "org.iso.mpeg.mxm.engine.securityengine.certificate.duration" [static]	
final String org.iso.mpeg.mxm.engine.securityengine.SecurityEngineKeys.COMMERCIAL_ENTITY = "org.iso.mpeg.mxm.engine.securityengine.commercial.entity" [static]	
final String org.iso.mpeg.mxm.engine.securityengine.SecurityEngineKeys.COUNTRY = "org.iso.mpeg.mxm.engine.securityengine.country" [static]	
final String org.iso.mpeg.mxm.engine.securityengine.SecurityEngineKeys.COUNTRY_NAMESPACE_URI = "urn:mpeg:mpeg21:2003:01-REL-SX-NS:country" [static]	
final String org.iso.mpeg.mxm.engine.securityengine.SecurityEngineKeys.KEY_STORE_PATH = "org.iso.mpeg.mxm.engine.securityengine.key.store.path" [static]	
final String org.iso.mpeg.mxm.engine.securityengine.SecurityEngineKeys.ORG_NAME = "org.iso.mpeg.mxm.engine.securityengine.org.name" [static]	
final String org.iso.mpeg.mxm.engine.securityengine.SecurityEngineKeys.ORG_UNIT = "org.iso.mpeg.mxm.engine.securityengine.org.unit" [static]	
final String org.iso.mpeg.mxm.engine.securityengine.SecurityEngineKeys.REGION = "org.iso.mpeg.mxm.engine.securityengine.region" [static]	
final String org.iso.mpeg.mxm.engine.securityengine.SecurityEngineKeys.REGION_NAMESPACE_URI = "urn:mpeg:mpeg21:2003:01-REL-SX-NS:region" [static]	
final String org.iso.mpeg.mxm.engine.securityengine.SecurityEngineKeys.SECURE_INFO_STORAGE_PATH = "org.iso.mpeg.mxm.engine.securityengine.secure.info.storage.path" [static]	
final String org.iso.mpeg.mxm.engine.securityengine.SecurityEngineKeys.SECURE_LICENSE_STORAGE_PATH = "org.iso.mpeg.mxm.engine.securityengine.secure.license.storage.path" [static]	
final String org.iso.mpeg.mxm.engine.securityengine.SecurityEngineKeys.SIGNATURE_ALGORITHM = "org.iso.mpeg.mxm.engine.securityengine.signature.algorithm" [static]	
final String org.iso.mpeg.mxm.engine.securityengine.SecurityEngineKeys.SYMMETRIC_ENCRYPTION_ALGORITHM = "org.iso.mpeg.mxm.engine.securityengine.symmetric.encryption.algorithm" [static]	

final String org.iso.mpeg.mxm.engine.securityengine.SecurityEngineKeys.SYMMETRIC_ENCRYPTION_KEY_LENGTH = "org.iso.mpeg.mxm.engine.securityengine.symmetric.encryption.key.length" [static]	
final String org.iso.mpeg.mxm.engine.securityengine.SecurityEngineKeys.USER_EMAIL = "org.iso.mpeg.mxm.engine.securityengine.user.email" [static]	
final String org.iso.mpeg.mxm.engine.securityengine.SecurityEngineKeys.USER_FIRST_NAME = "org.iso.mpeg.mxm.engine.securityengine.user.first.name" [static]	
final String org.iso.mpeg.mxm.engine.securityengine.SecurityEngineKeys.USER_LAST_NAME = "org.iso.mpeg.mxm.engine.securityengine.user.last.name" [static]	

9.11.7 SecurityInfoCreatorInterface

Classes implementing this interface are responsible for creating security-related data structures defined by W3C Digital Signature specification.

Operation	Description
EncryptedDataCreator org.iso.mpeg.mxm.engine.securityengine.SecurityInfoCreator.getEncryptedDataCreator ( ) throws SecurityEngineException	
EncryptedKeyCreator org.iso.mpeg.mxm.engine.securityengine.SecurityInfoCreator.getEncryptedKeyCreator ( ) throws SecurityEngineException	
KeyInfoCreator org.iso.mpeg.mxm.engine.securityengine.SecurityInfoCreator.getKeyInfoCreator ( ) throws SecurityEngineException	
KeyValueCreator org.iso.mpeg.mxm.engine.securityengine.SecurityInfoCreator.getKeyValueCreator ( ) throws SecurityEngineException	
SignatureCreator org.iso.mpeg.mxm.engine.securityengine.SecurityInfoCreator.getSignatureCreator ( ) throws SecurityEngineException	
X509DataCreator org.iso.mpeg.mxm.engine.securityengine.SecurityInfoCreator.getX509DataCreator ( ) throws SecurityEngineException	

9.11.8 SecurityInfoParserInterface

Classes implementing this interface are responsible for creating security-related data structures defined by W3C Digital Signature specification.

Operation	Description
EncryptedDataParser org.iso.mpeg.mxm.engine.securityengine.SecurityInfoParser.getEncryptedDataParser ( ) throws SecurityEngineException	
EncryptedKeyParser org.iso.mpeg.mxm.engine.securityengine.SecurityInfoParser.getEncryptedKeyParser ( ) throws SecurityEngineException	
KeyInfoParser	

org.iso.mpeg.mxm.engine.securityengine.SecurityInfoParser.getKeyInfoParser ( ) throws SecurityEngineException	
KeyValueParser org.iso.mpeg.mxm.engine.securityengine.SecurityInfoParser.getKeyValueParser ( ) throws SecurityEngineException	
SignatureParser org.iso.mpeg.mxm.engine.securityengine.SecurityInfoParser.getSignatureParser ( ) throws SecurityEngineException	
X509DataParser org.iso.mpeg.mxm.engine.securityengine.SecurityInfoParser.getX509DataParser ( ) throws SecurityEngineException	

10 ANNEX B – Survey of solutions of real world descriptors

In terms of real world descriptors we have a few solutions at our disposal nowadays. From existing solutions there are two of them that stand out: NRF ARTS and Etilize.

The Association for Retail Technology Standards (ARTS) of the National Retail Federation is an international membership organization dedicated to reducing the costs of technology through standards. Since 1993, ARTS has been delivering application standards exclusively to the retail industry. ARTS has four standards: The Standard Relational Data Model, UnifiedPOS, XML, and the Standard RFPs (in partnership with NRF). Membership is open to all members of the international technology community, retailers from all industry segments, application developers and hardware companies.

The ARTS Retail Data Model offers two distinct perspectives across the retail business. These are the Enterprise Context, which offers insight into the retail enterprise via three levels within the retail operation (home office, distribution and store levels), and Subject Area Composition, which gives an insight into the retail enterprise via the subject areas which cut across all three levels of retail operation.

The ARTS Retail Data Model currently supports ten of eleven retail business areas. These areas include:

- Merchandise flow management
- Inventory management
- Item and price maintenance
- Point of sale processing
- Tender control
- Store administration
- Customer relationship management
- Sales and productivity reporting
- Ordering (partially supported)
- Workforce Management (partially supported)

Etilize Inc. is a subsidiary of GfK Group founded in 2000 and is the largest product data provider of technology and office supply products in the world with databases on 7 million products in 30 countries and in 20 languages. Etilize provides product data to customers in more countries than any other provider of Information Technology and Consumer Electronics content. It offers product data information, product content and product data services for Resellers, Solution Providers, Comparison

Shopping Engines, Online E-tailers, E-commerce websites, Value-Added-Resellers, Manufacturers and Enterprise customers.

They publish standardized, e-commerce and retail store-ready product data across multiple industries:

- Information Technology
- Consumer Electronics
- Household Appliances
- Photography
- Home&Garden/ Do-It-Yourself
- Data Capture & Point of Sale
- Office Supplies
- Telecommunications
- Custom SKU Development to fit individual needs.