Project Number:	**FP7-257123**
Project Title:	**CONVERGENCE**
Deliverable Type:	**Report**
Dissemination Level:	**Public**
Deliverable Number:	**D6.1**
Contractual Date of Delivery to the CEC:	**31.10.2011**
Actual Date of Delivery to the CEC:	**24.12.2011**
Title of Deliverable:	**Network and Middleware Design**
Workpackage contributing to the Deliverable:	**6**
Nature of the Deliverable:	**Report**
Editor:	**Angelos-Christos Anadiotis (ICCS), Charalampos Patrikakis (ICCS) and Iakovos Venieris (ICCS)**
Author(s):	**Angelos-Christos Anadiotis (ICCS), Aziz Mousas (ICCS), Charalampos Patrikakis (ICCS), Dimitra Kaklamani (ICCS), Andrea Detti (CNIT), Matteo Pomposini (CNIT), Stefano Salsano (CNIT), Angelo Difino (CEDEO), Helder Castro (INESC), Maria-Teresa Andrade (INESC), Panagiotis Gkonis (SIL), Thomas Huebner (MORPHO), Andreas Kohlos (MORPHO)**
Abstract:	**This deliverable provides technical specifications for the implementation of the CONVERGENCE middleware and network protocols and technologies**
Keyword List:	**Middleware, network, MPEG-M, MXM, implementation, middleware, network protocols, specifications**

Executive Summary

Deliverable D6.1: “Network and Middleware Design” describes design principles and definitions to be used in the reference implementation of the CONVERGENCE network and middleware.

The deliverable is made up of two main sections:

1. The first section addresses issues related to middleware and middleware components, including issues regarding the implementation of the CONVERGENCE protocols, technology engines, aggregator and orchestrator engines. Since the CONVERGENCE middleware is derived from MPEG-M, we describe each component with reference to MPEG-M, highlighting CONVERGENCE contributions. The description of the middleware implementation begins with the general CONVERGENCE architecture, highlighting the role of protocol engines in the communication between peers. It then proceeds to describe the functionality provided by the generic components of the middleware and concludes with a description of the implementation of the protocols, technology and security engines and modules.
2. The second section addresses issues related to content-centric network design and implementation. The deliverable describes the design strategies, networking principles, and software architectures, concluding with a discussion of implementation issues.

A final section describes the technologies, protocols, standards, operating systems, development platforms and tools that will be used in the implementation work.

The deliverable concludes with a discussion of open issues that may affect our work.

We note that the deliverable is late with respect to its due date; however, this delay allowed us to: i) present a demo of the Convergence system at the 98th ISO/IEC JTC 1/SC 29/WG 11 (MPEG) Meeting, 2011/11/28-2011/12/02, Geneva Switzerland; ii) gather the related feedback and agree with MPEG-M the elementary services/protocol engines implementation; this was a necessary step since we want to conform with the MPEG-M standard; iii) prepare a CONVERGENCE Peer Kit (CPK) conformant to MPEG-M, which will be released right after this deliverable; iv), last but not least, take into account the comments of the first year review, which we received on 6/12/2011, especially those regarding the CoNet.

INDEX

1 Introduction

Work Package 6 of CONVERGENCE will provide open source reference implementations of the CONVERGENCE network and middleware, based on the architecture defined in WP3 and the protocols defined in WP5. The implementation provided by WP6 will be offered as a contribution to MXM.

Given that CONVERGENCE aims to re-use and extend MXM, the deliverable will frequently refer to the design and implementation of the MPEG-M middleware, highlighting specific CONVERGENCE contributions. In view of the general goals of the work package the deliverable will also discuss practical issues, such as the use of existing implementation tools and platforms, and their respective limitations.

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 Service (CDS)	A CoMid Technology Engine that provides all the matching concepts in a 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 Middleware Design and Implementation

3.1 Middleware Components

As previously described in deliverables D3.2 [2] and D5.1 [1] the fundamental unit in the CONVERGENCE middleware is the engine. There are two main types of engine: protocol engines and technology engines. The former parse protocols and call a chain of technology engines; the latter handle the atomic operations needed to perform specific actions. Each elementary service provided by the middleware is associated with a protocol engine (see D5.1).

Protocol and technology engines perform very specific operations. To orchestrate these operations we have introduced two additional classes of engines, the aggregators and the orchestrators. Their role is to combine protocol engines (the aggregator) and technology engines (the orchestrator) exposing a general protocol/interface that provides applications with a high level view of complex lower-level operations. For example, if the VDI creation process for an application includes content creation and identification, we can create a new Create & Identify Content engine that calls Create Content and Identify Content in sequence. We can then associate the engine with a new protocol that takes the required information as input, sets up the chain of engines (in this case Create Content and Identify Content) and returns the response defined in the aggregated protocol. In this sense, the aggregation introduces a new, custom, protocol that consists of the fundamental COMID engines building blocks. Then, the implementation of such an engine includes the implementation of a protocol engine used for this new custom protocol.

Each protocol engine has two parts: the client and the server part, each one usually implemented in different peers[1]. The general communication architecture between two CONVEVRGENCE peers, implementing the client and the server part of a protocol engine, is depicted in Figure 1.

[1] Even though the client and the server part could coexist in the same peer, this would be redundant, since they are implementing the same interface (more details are given in Section 3.4). Hence, without any loss of generality, we are going to assume that the client and the server part are running in different peers.

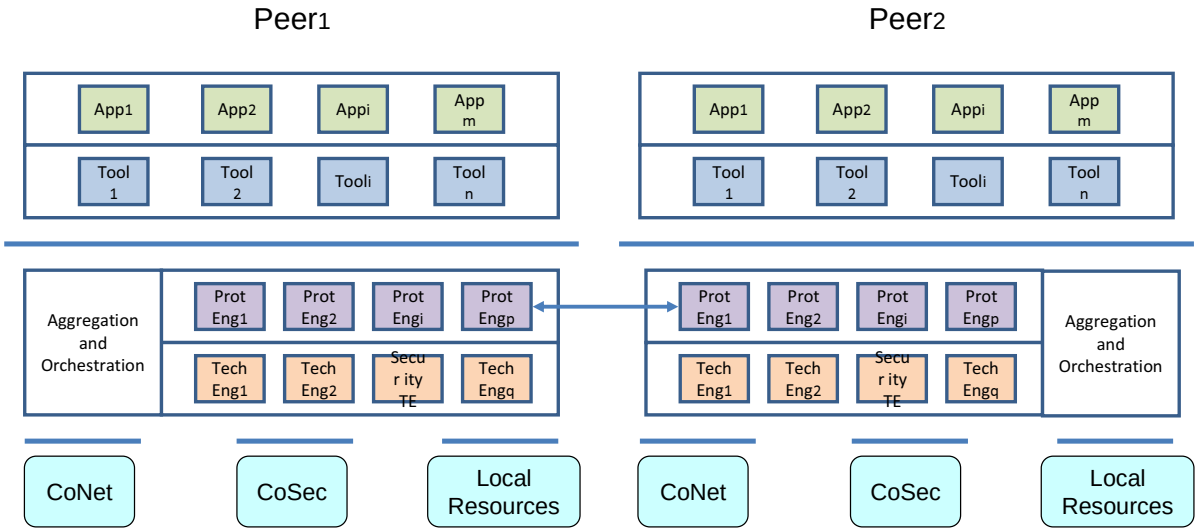

Figure 1 - Peers communicating with PEs

3.2 Abstract Middleware Design

CONVERGENCE middleware is based on the MPEG-M standard and, particularly, its middleware. The MPEG-M Architecture is specified in MPEG-M Part 1 – Architecture , the APIs are specified in Part 2 – MPEG extensible middleware (MXM) API - [4] and the implementation is provided in MPEG-M Part 3 – Reference Software. CONVERGENCE has adopted MPEG-M's abstract architecture and is thus in a position to enhance and extend MXM engines, while keeping the modular and extensible structure of MXM. The following Sections describe the basic functionality of each generic component of the middleware. This includes work developed as part of MXM and, when the existing functionality does not fully meet CONVERGENCE requirements, specific CONVERGENCE extensions to the standard. Several of these extensions have already been submitted as contributions to MPEG.

3.2.1 MXM/CONVERGENCE Core

The core is the part of MXM that guarantees interoperability between different middleware platforms and guarantees that they conform to the standard. To achieve the first of these goals, the core provides a general container that carries the data units and defines all the interfaces provided by MPEG-M APIs. This way, the access to technology engines of different platforms is made in a transparent way through the invocation of methods defined in the core interfaces, regardless of their implementation details. Middleware conformity with the standard is guaranteed by the core interfaces: if a middleware instance does not conform to the standard (i.e. if it does not implement the standard interfaces), applications will not be able to use it.

MXM defines a fundamental interface called *MXMObject* and a class implementing the interface (*MXMAdpater).* The interface and the class define basic containers for data passed between engines, ensuring interoperability between different middleware implementations.

Every information unit handled by the system has to be wrapped inside an MXMObject before it is passed to another engine. The second engine then converts it back to a data structure that it can handle.

Another important part of the core is the *MXMEngine* related classes and interfaces. These include:

- *MXMEngine:* every engine definition in the core extends this abstract class, which provides access to fundamental properties of MXM engines.
- *MXMEngineName:* this enumeration lists the name of all middleware engines.
- *MXMEngineResponse*: when an engine cannot return a specific result (e.g. an MXMObject) it returns an MXMEngineResponse: an enumeration that defines the following return types:
    - *SUCCESS*
    - *FAILURE*
    - *METHOD_NOT_IMPLEMENTED*

The core also defines the generic interfaces to be implemented by the classes responsible for handling specific portions of data in the engines (i.e. classes creating and parsing the data units exchanged by engines). A key examples is the *MXMCreatorAndParser* interface that extends the *MXMCreator* and the *MXMParser* interfaces.

The core is responsible for initializing middleware engines, checking if they exist and are operational and exposing them to the orchestrator and other higher level components. To discover engines, the core uses the *MXMConfiguration* file, which provides a mapping between engine names (hard coded in the *MXMEngineName* enumeration) and the class that implements the engine in a particular instance of the middleware.

Finally, the core contains the interface definitions (APIs) for the protocol and the technology engines, as specified in the MPEG-M standard and extended by CONVERGENCE.

The extensions of the standard proposed by CONVERGENCE can be seen as an initial validation of the MXM extensibility concept.

3.2.2 MXM/CONVERGENCE Data Object

This MXM module contains the schemas needed by the engines. Some of these schemas describe data structures that the engines have to deal with (for example the REL engine has to handle the REL XML schema); others represent the protocols used by the protocol engines (extracted from the corresponding elementary services defined in MPEG-M Part 4 and the CONVERGENCE deliverables produced byWP5).

In theory, it would be possible to define each engine's schema within the engine itself. In some cases, however, multiple engines use the same schema. Storing each schema inside a given engine (e.g. the REL schema in the REL engine), could create inter-dependencies between engines, jeopardizing the modularity of the middleware. We therefore decided to store schema in a dedicated module. This approach creates only one dependency: namely the

dependency on the general module. Given that this module contains the XML schemas for the standards involved in the middleware, and that these are bound to classes, it is expected that it will change only rarely.

3.2.3 MXM/CONVERGENCE Engines

This middleware component contains implementations of all the engines, including the aggregator and the orchestrator engine. Each engine implements the standard (or CONVERGENCE-specific) interfaces defined in the core. As noted previously, there should be no interdependencies between engines. If there is the need to combine operations provided by different engines, this should be accomplished via the orchestrator or the aggregator.

The orchestrator and the aggregator do not have a standard interface. However, they provide developers with the flexibility they need to define their own chains of engines, according to the requirements of the higher level components (tools/applications) they have to support.

3.3 Elementary Services Design and Implementation

CONVERGENCE implements elementary services as web services. The messages exchanged between an elementary service and the user/application are defined in MPEG-M Part 4 [6] and D5.1 [1]. This definition is used directly in the implementation, which creates request and response messages based on the schemas defined in these documents. This approach guarantees that the CONVERGENCE middleware implementation conforms to the MPEG-M standard and to CONVERGENCE protocols.

The implementation of elementary services is based on the protocol engines. In practice, the only role of elementary services themselves is to handle the communication between the server and the client; elementary service functionality is handled by the protocol engine, which makes the call to the orchestrator or directly to a technology engine, depending on what is required. More details are given in the next Section.

The aggregated services are also exposed as web services, adopting the same approach as for the elementary services, since they are essentially a combination of elementary services. No matter what this combination is, an aggregated service will have an input and an output protocol; and the communication part is handled by a web services framework.

3.4 Design and implementation of the Protocol Engines

The protocol engines are implemented both in the skeleton (server) and the stub (client) part of elementary services. On the skeleton side, they parse incoming requests, using the standard protocol schemas, subsequently invoking a chain of (technology) engines (or requesting an orchestration of technology engines) to perform the lower level operations, which produce the final result. This result is then formatted as an XML document, following the standard protocol schema, and sent back to the client. On the client side (stub) protocol engines are

responsible for issuing requests, for receiving and parsing the result and for delivering it to the application layer (e.g. a tool or the application itself).

Figure 2 provides an example of this approach, showing the implementation schema for the Create Content protocol engine and the corresponding elementary service. The abstract create- content-engine and the elementary service interface definition are in the core where they can be accessed by any part of the middleware. As it can be seen in the Figure, there are two possible implementations of the Create Content PE: the first (Ver. 1) calls the Content Creation Orchestration and creates the VDI; the second (Ver. 2) is just a web service client that uses the Create Content Elementary Service (ES) to make a call to the server side (i.e. Create Content PE ver. 1), which creates the VDI and returns it to the client. As it can be observed in the Figure, the two Create Content PE versions extend and implement the engine abstract class in the core. This means that it is possible to make a call to the protocol engine by just using the core API and, then, depending on the middleware implementation (ver.1 or ver.2), create the VDI either locally or remotely. The call and the result are the same in both cases. The MXM Configuration file registers the version used by the application and provides the mapping between the engine names and their implementations, thus ensuring that the implementation of the middleware is transparent to applications and tools. Developers do not have to be aware of the protocol engine implementation – they just use the core API [6].

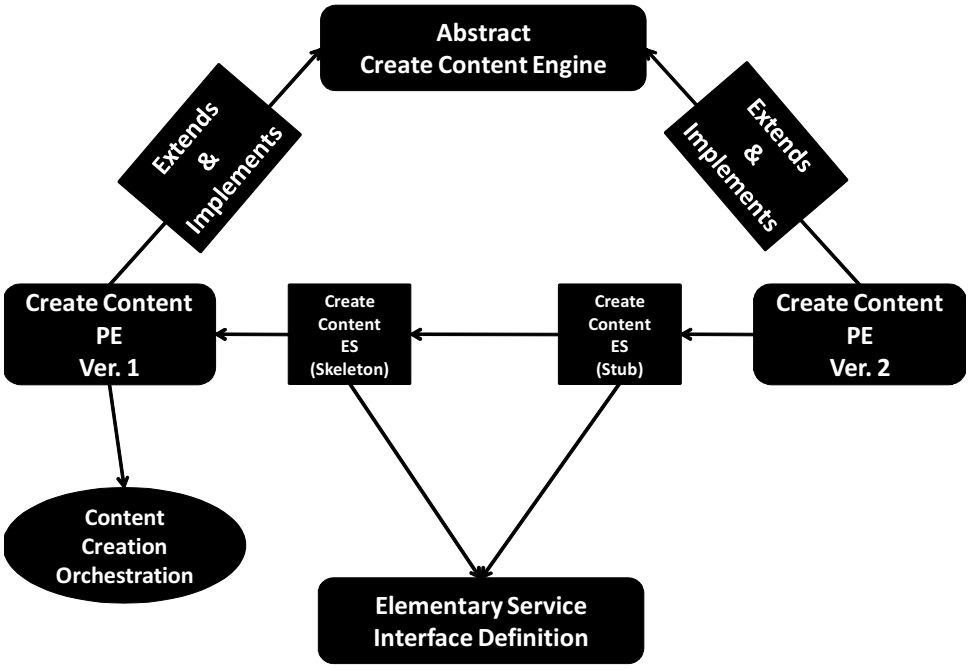

Figure 2–Implementation of Protocol Engines/Elementary Services

Aggregation of the elementary services is based on the same basic scheme; in essence, we just need one more protocol to handle communication between the aggregated service (seen as a

black box) and the upper layers (applications, tools). Everything else remains the same: the aggregator triggers protocols just like an application or a tool.

3.5 Design and implementation of the Technology Engines

Technology engines are the lower level components of the CONVERGENCE middleware; they are responsible for managing the available technology to perform the operations defined through their standard API. Each Technology Engine consists of interfaces, classes, methods (see D3.2 [2]) which can be called either directly by application layer components or by other middleware components (Protocol Engines, and Orchestrators). A Technology Engine is never called by another Technology Engine.

3.6 Design and implementation of the Security Module

The CONVERGENCE Security Module (CoSEC) uses a distributed architecture to handle cryptographic protocols involving two (or more) partners. CoSEC comprises interacting software components installed on client's PCs, server PCs and smart cards.

Each user is equipped with a smart card containing cryptographic keys for:

1. Authentication (Challenge-Response)
2. (VDI-) Signatures
3. Decryption (usually hybrid decryption of content/VDIs)

Smart Cards use their own standardized communication protocols (APDUs). Java libraries created in CONVERGENCE will completely encapsulate any communication with the smart card, so that other components need not be aware of smart card related standards.

Java tools will provide functionality for:

- Identification (= Registration)
- Authentication
- Signing / Verifying
- Decryption/Encryption

The trust relationship between different entities (Identity Providers, Service Providers, CONVERGENCE applications, CONVERGENCE clients etc.) is based on a certificate structure using X.509 certificates.

4 Network Design and Implementation

As discussed in [1] and shown in Figure 3, CONVERGENCE will provide two different implementations of CONVERGENCE network (CONET) API. In track 1, we will use a CONET Technology Engine (TE) that operates on top of a plain TCP/IP network, providing the CONET service at the middleware level. In this case there will be no need for network design or implementation. In track 2, the CONET TE will operate on top of a novel content-centric network layer designed and implemented by CONVERGENCE. It is this content-centric network we will discuss in what follows.

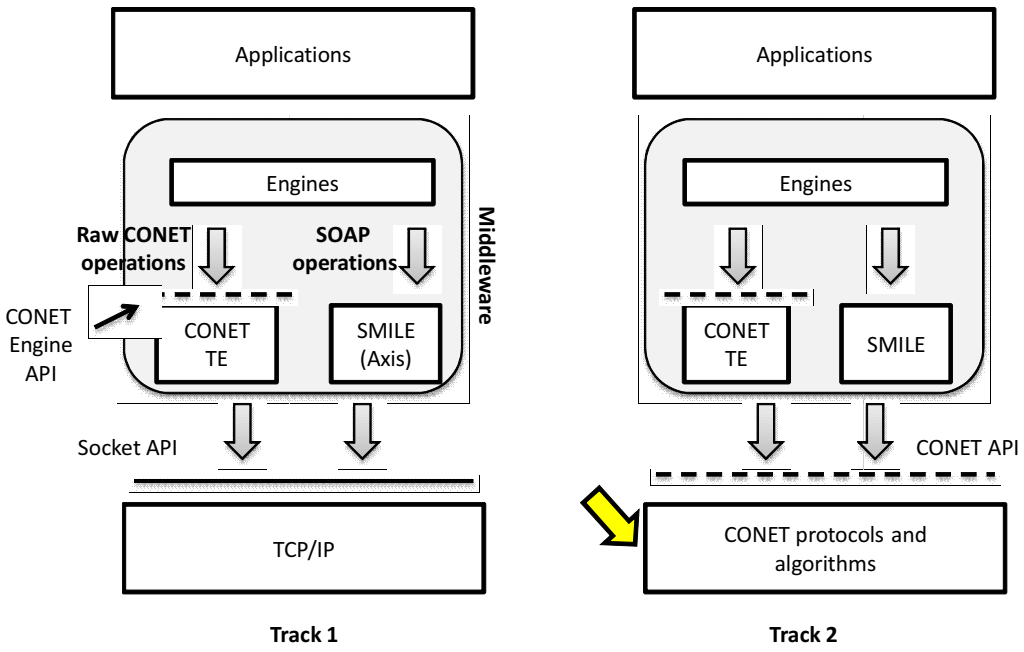

Figure 3 - CONET services deployment strategy

4.1 Design strategy

Since the CONET network architecture is derived from the CCNx architecture [7] proposed in [8], we choose to base our software on a modified version of the CCNx software.

The CCNx package runs both on Linux and on Android. This feature will facilitate CONET deployment on mobile devices.

In what follows we will begin by describing CCNx networking principles and the CCNx software. We will then go on to identify the differences between CCNx and CONET and discuss the issues they raise for the CONVERGENCE implementation.

4.2 CCNx networking principles

To maintain consistency with the CONET terminology introduced in D3.2[2], we have mapped CONET terminology to CCNx terminology. In this mapping the CCNx "Data packet" is equivalent to the CONET named-data CIU, the CCNx "Interest packet" is equivalent to the CONET Interest CIU and the CCNx string (name/chunk-number) that identifies a CCNx Data packet is equivalent to the CONET Network Identifier (NID). For a full description of CCNx we refer to [7] for the software and to [8] for networking principles.

CCNx provides users with the possibility of downloading a named-data CIU by specifying its name and chunk number, that is its Network IDentifier (NID). To download a named-data CIU, a CCNx end-node sends out an Interest CIU that contains the NID of the requested named-data CIU. Network nodes forward the Interest CIU towards the best serving-node. Forwarding is based on the NID contained in the Interest CIU (content-routing) and a name-based forwarding table, called the Forwarding Information Base (FIB). Entries in the FIB have the form <name-prefix, next-hop address/port>. Interest CIUs are forwarded via a plain UDP/IP socket, whose remote endpoint is the next hop CCN node.

Every time a node forwards an Interest CIU, the node registers the requested NID and the identifier of the ingress interface in a table called Pending Interest Table (PIT). This scheme enables CCNx to support reverse routing (i.e. from serving-node to end-node).

When the Interest CIU reaches the serving-node or a node that has cached the requested named-data CIU, the node retrieves the data item from its own Content Storage (CS) and sends back the named-data CIU. The named-data CIU is forwarded toward the end-node by exploiting and consuming the information in the PITs of the nodes the Interest CIU traversed previously. Every time a node receives a named-data CIU, the node looks up the output interface in the PIT, uses the interface to forward the named-data and deletes the PIT entry. Furthermore, when a node receives a named-data CIU, it can cache the data item in its Content Storage (CS), thereby enabling in-network/en-route caching.

Forwarding of CIUs and in-network caching should be considered as the "core" functions of CCNx, which the software supplements with other "high-level" functions, allowing users to download a complete named-data, and to manage Content Storage.

The downloading function ("getfile") implements a receiver-driven congestion control mechanism, in which the end-node sends the complete sequence of Interest CIUs required to download the set of named-data CIUs, forming a whole named-data[2].

The Content Storage management function enables users to create named-data CIUs, starting from a whole named-data, and to store them in the Content Storage itself.

[2]As described in [2], a named-data CIU is only one chunk of a complete named-data (e.g. a file).

4.3 CCNx software architecture and functionality

The CCNx software consists of core functions implemented by a *ccnd* daemon written in C language and higher-level functions, mainly implemented as Java applications.

The ccnd daemon manages the FIB, the PIT and CS. When an Interest CIU arrives, the daemon checks the Content Storage (CS). If the CS contains a named-data CIU that matches the Interest CIU, the daemon responds to the Interest CIU with a named-data CIU. Otherwise, the daemon extracts the address of the next hop node toward the serving-node from the FIB and creates an entry in the PIT to support reverse forwarding of the named-data CIU. If there is no matching entry in the FIB, the daemon discards the Interest CIU.

4.4 Key issues for the CONET implementation

The main issues for the CONET implementation arise from the need to implement a number of functions that are not present in CCNx:

1) *Lookup-and-cache routing architecture*. CCNx does not have a mechanism to limit the size of the name-based routing table. This means that with 10^{11} name-prefixes (as we expect), we would need a FIB with 10^{11} entries. Such a large forwarding table is not a technologically feasible solution for forwarding packages over high speed links. To achieve fast lookup, indeed, forwarding tables are realized by TCAM or SRAM memories that have a very limited space. To "alleviate" this *scalability* problem, we have devised a lookup-and-cache routing architecture, in which the FIB contains only the subset of routes currently used to forward traffic. In this scheme, missing entries in the FIB are retrieved from a remote RIB (Routing Information Base) hosted by a Name Routing System (NRS) node, and inserted into the FIB where, if necessary, they replace routes considered inactive. The route replacement strategy has a relevant impact on performance, and we devised and implemented the Inactivity Time Out (ITO) replacement strategy (better described in the forthcoming D5.2). ITO infers which are the inactive routes and in case replaces them, rather than replacing active routes. Furthermore ITO operates without pre-emption, i.e. an active route can never be replaced. For the sake of comparison, we also implemented the well-know Least Recently Used (LRU) replacement policy.
2) *NRS routing-layer*. The routing protocol is aimed at disseminating the routing entries among NRS nodes. We need to implement this routing-plane from scratch, indeed CCNx does not provides a routing protocol.
3) *Carrier-packets*. CCNx directly transfers Interest CIU and named-data CIU in large IP/UDP packets. To reduce security overhead, CCNx uses large named-data CIU, with sizes in the order of tens of kilo bytes (up to 8kB). These large CIUs require IP segmentation. Therefore, each CCNx node has the burden of reassembling an IP packet before it can forward a named-data-CIU. To overcome this issue and to further reduce security overhead, CONET introduces named carrier-packets, low-level

carriers of CIUs. A carrier-packet is an IP packet with the IP CONET option. Using carrier-packets, the CONET could use a very large named-data CIU (e.g. greater than 65 kilo bytes), so limiting security overhead without the need of reassembly data units at each CONET node.

4) *Source-routing*. CCNx nodes are stateful, in the sense that the PIT stores status information on ongoing communications. CONET nodes are stateless. What makes this possible is the source-routing approach used to route data back from the end-node to the serving-node. Every time a border-node forwards an Interest CIU, it appends an identifier of the output interface to the data unit. This gives the serving-node a list of the interfaces the Interest CIU has traversed, enabling it and next nodes to return data to the end-node.
5) *Congestion-control*. CCNx perform congestion-control on named-data CIUs. However, we argue that its large data-units reduce the performance of the congestion control algorithm. Therefore, CONET performs congestion control on carrier-packets. This approach should provide performance similar to that of TCP.
6) *Internal node*. CCNx does not have this type of node. Internal nodes perform forwarding operation by means of IP, but are able to cache CONET named-data CIUs.
7) *Named-sap*. CCNx provides users with the possibility of fetching named-data. The CONET extends the scope of the service by allowing users to push information to a remote server identified by a name (i.e. the named-sap).

4.5 CONET implementation

We plan to extend the CCNx software, handling the seven issues identified in the previous section. At the time of writing, implementation of the forwarding-plane of the lookup-and-cache routing architecture (i.e. the first issue) is complete. In our laboratories we have a preliminary and running version of CONET border-nodes, end-nodes, serving-nodes and NRS nodes.

Figure 4 reports the modules involved in the lookup-and-cache forwarding operation. We introduced a number of modifications to the ccnd daemon. Now, if the FIB does not contain an entry for an incoming Interest CIU, the ccnd generates a request to a local lookup-and-cache server, including the name (NID) of the named-data CIU the Interest CIU refers to. The local lookup-and-cache server queries a Name Routing System (NRS) node, implemented using bind-9 DNS server software. The NRS node replies to the lookup-and-cache server with the IP address of the next border-node toward the serving-node. When the lookup-and-cache server receives the reply, it commits the ccnd to insert a new route in the FIB. If the FIB had reached the maximum allowed size, the insertion of the new route is controlled by the selected route replacement policy, i.e. LRU or ITO.

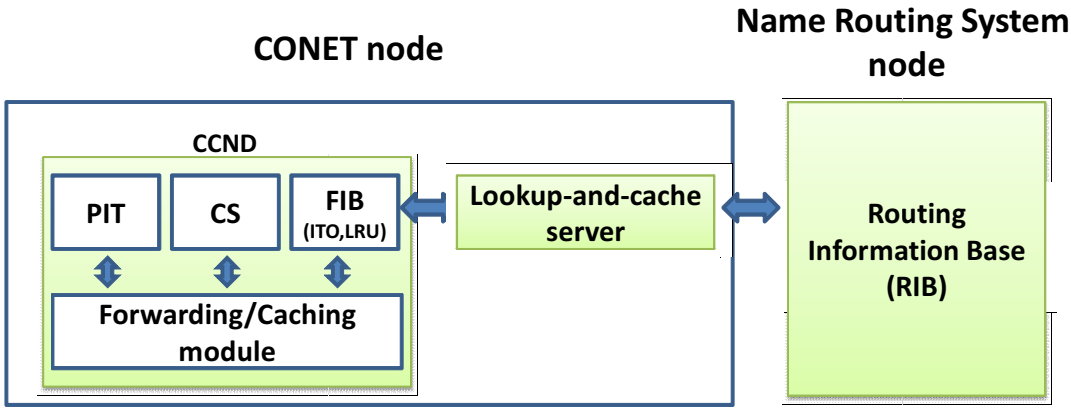

Figure 4 - Implementation of lookup-and-cache forwarding plane

5 Implementation Technologies

5.1 Standards

The CONVERGENCE middleware (CoMid) is based on and extends the standard MPEG middleware (MPEG-M). Especially for implementation, CoMid follows the guidelines of the MPEG eXtensible Middleware (MXM) standard implementation (MPEG-M Part 3 – Reference Software [5]), while simultaneously contributing new components needed in both projects.

As far as concerns authentication and security and the underlying Public Key Infrastructure (PKI), CONVERGENCE will use established standards such as X.509 certificates and PKCS signature schemes. Given the lack of established international standards the "Restricted Identification" feature will be based on the German nPA standard (used for electronic passports). CONVERGENCE Smartcard technology will be based on the ISO 7816 standards.

5.2 Devices

The CONVERGENCE middleware has been successfully tested on PC and MAC workstations and laptops, running Java Virtual Machine (JVM) version 5 and higher. It has not yet been tested on portable mobile devices (such as mobile phones or tablets). These tests are planned for the near future. CONVERGENCE's network test-bed currently operates successfully on laptops and workstations.

5.3 Operating Systems

The components of the CONVERGENCE middleware have been successfully tested on Windows Vista; Windows 7; Linux; and MacOS X. Nonetheless, as components are being developed in Java, they are expected to run successfully on any operating system which supports Java standard edition. In the near future, it is planned to integrate CONVERGENCE middleware with the Android operating system.

The CONVERGENCE network operates on the Linux operating system. Work is in progress to run the network over Android.

5.4 Protocols

The CONVERGENCE middleware relies on the following underlining protocols:

- HTTP/SOAP. The elementary services that make up CONVERGENCE middleware are implemented as web services. Web services communicate using the HTTP and SOAP protocols.

- SIP. In track 1 of the CONVERGENCE project, the “advertising “and “sending to a named service-access-point” operations on the CONET TE are implemented using the SIP protocol. In track 2, these operations will use the CONVERGENCE network API.

The CONVERGENCE network relies on the CONET protocols specified in [1] and on IP.

5.5 Servers

CONVERGENCE provides two distinct implementations of Elementary services:

- Using the EJB3 framework based on the HTTP protocol (see Section 6.2);
- Using the sendToName primitive in the CONET (see Section 5.4).

In the first implementation the application server will be Apache Tomcat 6 with EJB3 support provided by the Apache OpenEJB3 library. Elementary services will also be tested with JBoss v5.1.

The second implementation does not require the use of a server.

6 Development Platforms

6.1 Languages

The CONVERGENCE middleware components are written in Java (Java Standard Edition, version 5 and higher), which guarantees interoperability across different operating systems and is supported by a large variety of frameworks, including application servers. The CONVERGENCE network is written in C language and in Java (Java Standard Edition, version 5 and higher).

6.2 Tools and external Frameworks

Development work on the CoMid is based on the following tools:

- Apache Maven

  The maven project structure provides a very simple, yet efficient way to organize and maintain complex projects such the CONVERGENCE middleware. The organization of the project is similar to that of MXM.

  The root maven project ("middleware") contains the following modules:
  - convergence-core – this module extends the mxm-core with standard interfaces and abstract classes;
  - convergence-engines – this module consists of multiple sub-modules, each consisting of an engine;
  - convergence-apps – this module includes basic applications, used to demonstrate middleware functionality. User interfaces are deliberately simple.
- Java Architecture for XML Binding (JAXB)

  The JAXB framework is used:
  - to transform the XML schemas for protocols and data structures (REL statements, VDIs, etc.) into Java classes
  - to *marshall* (go from XML document to Java object) and *unmarshall* (go from Java object to XML document) dynamically in the code
- Enterprise Java Beans 3.0 (EJB3)

  The EJB3 framework is used to implement elementary services. The interfaces for elementary services use EJB3 annotations; remote calls to services use the JAX-WS 2.0 framework
- jBPM (Business Process Management)

  We are currently experimenting with jBPM to support the dynamic execution of BPMN version 2.0 processes. This would allow us to use BPMN to define and execute complex aggregation workflows [10].
- Eclipse Toolset

The Eclipse Maven and web tools platform (WTP) plugins are used to facilitate the development process. The Drools and BPMN2 plugins provide graphical BPMN editors.

7 Limitations and open issues

Even though the network and middleware design described in this deliverable provides clear guidelines, the actual implementation of the engines used in CONVERGENCE will have to take account of the limitations imposed by specific operating systems, operating system versions and devices. In what follows, we will discuss the most important of these issues and describe proposed solutions.

7.1 Mobile Devices

Mobile devices are gaining ground among today's Internet users. In recognition of this fact, CONVERGENCE will shortly run tests to assess whether CONVERGENCE's middleware and network implementations can be deployed on the Android platform. The choice of Android is motivated by the open nature of the OS, the possibility of deploying of applications through a direct download and install process, and the similarities with open source OS implementations such as Linux. However, it is still not sure whether the deployment will be possible. If the results of the tests prove to be negative we will report the problems we have encountered and investigate possible solutions.

7.2 Development Languages

As mentioned earlier, our current implementation of the CONVERGENCE middleware uses the Java programming language. However not all devices support Java. Supporting these devices would require a completely new implementation of the middleware. At this stage, therefore, CONVERGENCE will be restricted to mobile devices which support Java.

7.3 Orchestration and Aggregation

CONVERGENCE requires a unified approach to aggregation that exploits standard APIs and protocols. Work on this issue is ongoing. As mentioned in Section 6.2, one option currently under investigation is to use the jBPM framework to host the execution of chains of protocol engines. However, this would require a lot of work to define APIs and elementary services and a framework providing full support for the execution of BPMN v2.0 workflows, which is currently lacking.

In the absence of such a framework we will use BPMN to define service aggregation, as specified in MPEG-M Part 5 [11] but will not directly execute BPMN-defined workflows. In this case aggregation will be implemented in the same way as other elementary services.

7.4 Security

CONVERGENCE security software will be implemented using the following software:

1. Java Standard Edition packages (security packages), including the sun.security.* package for standard cryptographic services like symmetric encryption, asymmetric cryptography for digital signatures and encryption/decryption with RSA or ECC (Elliptic Curve Cryptography).
2. Java Standard Edition for advanced cryptographic services supporting GS (Group Signature), IBE (Identity Based Encryption) and ABE (Attribute Based Encryption). This software will also support functionality envisaged for later phases of the project.
3. Dedicated Smart Card software used on-card: cryptographic library functionality specialized for the Smart Card IC at hand, OS (operating system) on-card.

In implementing points 1 and 2, we will attempt to avoid software not included in standard Java. However, more sophisticated security tasks may require the use of additional software such as “Bouncycastle”.

8 Bibliography

[1] A.-C. Anadiotis, C. Patrikakis, I. Venieris, “Requirements and Initial Protocol Architecture”, CONVERGENCE Public Deliverable D5.1, May 2011.

[2] M. Tanase, L. Corlan, S. Salsano, “System Architecture”, CONVERGENCE Public Deliverable D3.2, July 2011.

[3] ISO/IEC 23006-4 – Information Technology – Multimedia Service Platform Technologies – Part 1 – Architecture

[4] ISO/IEC 23006-4 – Information Technology – Multimedia Service Platform Technologies – Part 2 – MPEG extensible middleware (MXM) API

[5] ISO/IEC 23006-4 – Information Technology – Multimedia Service Platform Technologies – Part 3 – Reference software and Conformance

[6] ISO/IEC 23006-4 – Information Technology – Multimedia Service Platform Technologies – Part 4 – Elementary Services

[7] Content centric Networking reference software, CCNx, available at http://www.ccnx.org/

[8] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, R. L. Braynard, “Networking Named Content”, ACM CoNEXT 2009, Rome, December, 2009.

[9] A.-C. Anadiotis, A. Difino, I. S. Venieris, “Proposed PE/ES Implementation”, Input contribution for MPEG-M Part 3, Geneva, November 2011.

[10] A.-C. Anadiotis, A. Difino, I. S. Venieris, “First steps towards a unified aggregation/orchestration environment”, Input contribution for MPEG-M Part 5, Geneva, November 2011.

[11] ISO/IEC 23006-4 – Information Technology – Multimedia Service Platform Technologies – Part 5 – Service Aggregation

9 ANNEX A –Developer's Guide to COMID

9.1 Checkout the Source Code

Since CoMid is based on MXM, the first thing a developer needs to do is to synchronize with the MXM SVN repository:

```
svn co http://wg11.sc29.org/mxmsvn/repos/JAVA/trunk mpegm
```

This repository has public read access, using the following credentials:

- Username: mxmpubro
- Password: mpegmxmro

The second step is to check out the CoMid repository from the following location:

```
svn co https://minerva.netgroup.uniroma2.it/svn/convergence/SOFTWARE/middleware/CoMid/trunk middleware
```

Since the middleware is still under development, the CONVERGENCE repository is not yet public. The repository will be made public at the end of the implementation.

CONVERGENCE applications can also be found in the CONVERGENCE repository:

```
svn co https://minerva.netgroup.uniroma2.it/svn/convergence/SOFTWARE/CoApp/applications/trunk application
```

9.2 Source Code Structure

Figure 5 depicts the file structure of the MXM and CoMid repositories. Some engines are the same in both repositories while some are different. This is because in some cases CoMid uses MXM engines as is, or extends their functionality while maintaining the same name. In other cases CoMid introduces completely new engines.

Figure 5 - MXM and COMID File Structure

9.3 Build the Code

As already mentioned in Section 6.2, we use Apache Maven to setup and build the source code. In both cases (CoMid and MXM) we build/install the whole project from the top level pom.xml file. If we want to install only a specific engine, we can use its specific pom.xml file. The usual command to build and install our project is:

```
mvn clean install
```

Now that the code is built, installed and ready to be imported into Eclipse, any developer can start from this basis and extend CoMid or MXM.

To import the project to Eclipse, we suggest using the following command and then simply import the project to Eclipse as a Maven Project:

```
mvneclipse:eclipse
```

9.4 CoMid Engine Implementation Guide

To implement a custom engine<`CustomEngine`>, a developer should work in following steps:

1. Create the engine abstract class and interfaces in the core. For technology engines use package
   `eu.convergence.middleware.engine.<customEngine>TE.<CustomEngine>`
   For protocol engines use the package:
   `eu.convergence.middleware.engine.<customEngine>PE.<CustomEngine>`
   This engine extends the
   `org.iso.mpeg.mxm.core.MXMEngine`
   as described in Section 3.2.1.
2. Data-handling interfaces for the new engine should extend the `org.iso.mpeg.mxm.core.MXMCreatorAndParser` core interface.
3. If there are any exceptions related to the engine, they should be include in the package `eu.convergence.middleware.engine.<customEngine>TE.exception`
4. The <`CustomEngine`> name must be added to the
   `eu.convergence.middleware.core.ConvergenceEngineName`
   enumerator.
5. Create the new engine project as a module in the convergence-engines Maven project.
6. Implement the abstract classes and interfaces in the core.

9.5 MXM Configuration

CONVERGENCE uses the MXMConfiguration file notation to instantiate the engines required by an application. The MXMConfiguration.xml file looks like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<mxm:MXMConfigurationxmlns:mxm="org:iso:mpeg:mxm:configuration:schema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="org:iso:mpeg:mxm:configuration:schema mxmconfiguration.xsd">
...
<mxm:MXMEngine id="0" type="CreateContentPE">
<ClassName>
eu.convergence.middleware.createcontentPE.remote.CreateContentPE
</ClassName>
<EngineParameters>
<entry key="SERVICE_URL">
http://localhost:9090/Convergence_CreateContentES/CreateContent?wsdl
</entry>
<entry key="SERVICE_NAME_NS">
http://service.CreateContentES.middleware.convergence.eu/
</entry>
<entry key="SERVICE_NAME">
CreateContentESService
</entry>
</EngineParameters>
</mxm:MXMEngine>
...
</mxm:MXMConfiguration>
```

This file should be included in the application classpath, so that it can be passed to MXM. MXM will then instantiate all engines included in this configuration file.