Project Number:	**FP7-257123**
Project Title:	**CONVERGENCE**
Deliverable Type:	**Prototype**
Deliverable Type:	**Public**
Deliverable Number:	**D6.2**
Contractual Date of Delivery to the CEC:	**30.06.2012**
Actual Date of Delivery to the CEC:	**03.08.2012**
Title of Deliverable:	**Network and Middleware Implementation**
Workpackage contributing to the Deliverable:	**WP6**
Editors:	**Angelos-Christos Anadiotis, Charalampos Z. Patrikakis**
Author(s):	**Angelos-Christos Anadiotis, Aziz Mousas, Kostas Papadopoulos, Nikos Dellas, Georgios Lioudakis (ICCS), Andrea Detti, Matteo Pomposini (CNIT), Angelo Difino (CEDEO), Fernando Almeida (INESC PORTO), Thomas Hübner, Carsten Rust (MORHPO), Panagiotis Gkonis, (SIL) Mihai Tanase (UTI).**
Abstract:	**D6.2 is classified in the DoW as a prototype. The D6.2 prototype comprises the implementation of the CONVERGENCE middleware and network protocols.** **Following the completion of the D6.2 prototype, the consortium decided to issue this report, considered complementary to the prototype.** **The report, bearing the same name as the prototype, provides an introduction to (and the use of) the CONVERGENCE middleware and network for applications development.**
Keyword List:	**Middleware, network, guidelines, implementation, prototype, MXM, technology engines, protocol engines**

Executive Summary

D6.2 is classified in the DoW as a prototype. The D6.2 prototype comprises the implementation of the CONVERGENCE middleware and network protocols.

Following the completion of the D6.2 prototype, the consortium decided to issue this report.

This document is intended as a reference guide for developers, and should be seen as complementary to the prototype of the CONVERGENCE middleware and network protocols. The report provides all the information necessary to understand the implementation, together with useful links to deliverables from WP3, WP5 and to the MPEG standard.

INDEX

1 Introduction

Following the definition of the intermediate architecture in D5.2, the project implemented middleware and network protocols. The D6.2 prototype comprises such implementation. Following the completion of the D6.2 prototype, the consortium decided to issue this report, considered complementary to the prototype.

The report, bearing the same name as the prototype, provides an introduction to (and the use of) the CONVERGENCE middleware and network for applications development.

This document is mainly intended for developers. The goal is to provide them with the information necessary to:

a) Understand the implementation

b) Build applications on top of the implementation.

The document is organized in two sections, referring respectively to the middleware and the network.

2 Glossary

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.
CA	Central Authority
CCN	Content Centric Network
Cl_Auth_SC	Client Authentication with Smart Card (Challenge Response)
Cl_Auth_User_Pw	Client Authentication with Username and Password
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

Service (CDS)	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	The Content Centric component of the CONVERGENCE

Network (CoNet)	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".
Dec_Key_Unwrap	Key Unwrapping and Content Decryption
DIDL	Digital Item Description Language
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.
Enc_Key_Wrap	Encryption and Key Wrapping
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.
Group_Sig	Group Signature
ICN	Information Centric Network
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"
IP	Identity Provider
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 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.

Named-data	A named-resource consisting of data.
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"
PKI	Public Key Infrastructure
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 (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 (Rights Expression Language)	The User to whom Permissions are Granted in a License.
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.
REL	Rights Expression Language
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.).
Serv_Auth	Server Authentication without Smart Card
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.
Sig	Signature
Smart_Card Role_Auth_SC	Role Authentication towards Smart Card
SP	Service Provider
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.
Us_Reg_IP	User Registration to Identity Provider
Us_Reg_SP	User Registration to Service Provider

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 Implementation

3.1 Middleware Source Code Organization

CONVERGENCE middleware is developed in JAVA and organized in the same fashion as the MPEG-M middleware. CONVERGENCE use both engines that reside in MXM and CONVERGENCE's own engines. However, many of the latter engines have already been adopted by MPEG-M and have been developed with this goal in mind.

As illustrated in MPEG-M Part 3 [1], the MPEG-M middleware is a Maven project. The software consists of the core, the engines, the elementary services and the data object.

The core module includes the basic functionality needed for initializing and instantiating the middleware, the APIs for the protocol and for the technology engines and for the elementary services. The APIs included in the core reflect the specifications in MPEG-M Part 2 [2] and CONVERGENCE deliverable D5.2 [3].

The engines module provides a reference implementation of the engine APIs specified in the core.

The elementary services module contains the implementation of the elementary services specified in the core.

The dataobject module contains the implementation of the schema creators and parsers; in particular, the engines, in most cases, need to perform several actions related to the standard schemas, e.g. create a VDI, call an elementary service using a standard protocol etc.

CONVERGENCE and MXM use the Java Architecture for XML Binding (JAXB). However, the technology is designed to allow the use of different parsers to meet different needs. In particular, the dataobject module has been implemented to facilitate integration with a vast variety of other technologies, such as DOM or even plain String handling.

3.2 Implementation Guidelines

This section provides implementation guidelines for developers using the Convergence middleware for the first time.

CONVERGENCE is based on the MXM middleware. Thus, developing for CONVERGENCE/MXM can mean:

- Implementing a new CONVERGENCE/MXM Engine from scratch
- Extending an existing engine with new functionality
- Orchestrating CONVERGENCE/MXM Engines to perform a complex application-specific action.

In the first and second cases, it is necessary to have an in-depth knowledge of the MXM architecture (see "Developing an MXM Engine" in 3.2.2.3). In the third case, the developer

needs to be familiar with the normative API for the relevant MXM Engine (see “Using an MXM Engine” in 3.2.2.4).

3.2.1 Schemas

Most of the Engines implemented in the CONVERGENCE project use specific xml schemas to represent the information they manage and to guarantee interoperability with other engines. For example, when an MXM application developer wants to create a License to be embedded in a VDI, she will use the REL Engine to create a REL license and a VDI Engine to embed it in the VDI. A set of normative APIs provides abstract representations of the REL license and the way it is passed to the VDI engine. These are called "schema handler APIs". Another set of APIs allows engines to access Engine functionality. These are collected in a set of "technology handler APIs".

Below, we provide references to the schemas for different aspects of the MXM middleware.

DI Engine / Standard	Profile or Technology	Reference / Schemas' URI
Digital Item Declaration	Main MSAF	[ISO/IEC 21000-2] [ISO/IEC 23000-7]
Digital Item Identification		[ISO/IEC 21000-3]

MP21FFEngine / Standard	Profile or Technology	Reference
MPEG-21 File Format		[ISO/IEC 21000-9]

REL Engine / Standard	Profile or Technology	Reference
Rights Expression Language	MAM profile	[ISO/IEC 21000-5]
	DAC profile	[ISO/IEC 21000-5]
	OAC profile	[ISO/IEC 21000-5]

IPMP Engine / Standard	Profile or Technology	Reference
MPEG-21 IPMP Components	Main MSAF	[ISO/IEC 21000-4] [ISO/IEC 23000-5]
IPMP XML Messages		[ISO/IEC 23001-3]

Metadata Engine / Standard	Profile or Technology	Reference
MPEG-7 MDS	Version 3	[ISO/IEC 15938-5]

ER Engine / Standard	Profile or Technology	Reference
Event Reporting		[ISO/IEC 21000-15]

Security Engine / Standard	Profile or Technology	Reference
XML-Encryption Syntax and Processing		[XML-Encryption Syntax and Processing - W3C Recommendation 10]
XML-Signature Syntax and Processing		[XML-Signature Syntax and Processing - W3C Recommendation 12]
Trusted Software Stack		[Trusted Computing Group , "TCG Software Stack (TSS) Specification Version 1.2"]

Search Engine / Standard	Profile or Technology	Reference
MPEG Query Format		[ISO/IEC 15938-12]

3.2.2 Engine Implementation

The Engine APIs are grouped in two categories:

- "Schema handler APIs" represent the data managed by an Engine.
- "Technology handler APIs" allow applications to access the functionality provided by an engine.

3.2.2.1 Schema Handler

The set of APIs in charge of accessing/managing the data represented by a particular engine (i.e. a REL License or a Grant represented in the REL Engine) is called the schema handler.

Schema Handler APIs reflect the schema for the data managed by a particular engine, independently of how the data is managed (created, parsed) in any specific xml representation (DOM, Java binding classes, SAX, …etc.). The APIs are generated semi-automatically. The schema representation allows developers to use the schema, without any deep knowledge of

the schema hierarchy. As described earlier, the CONVERGENCE schema handlers use JAXB tools. JAXB creates simple JAVA utility classes that implement the APIs in a straightforward way.

3.2.2.2 Technology Handler

The Technology Handler APIs provide access to the functionality offered by a specific Engine. There is no standard representation for these APIs; engine developers specify handlers in the way that best suits their requirements.

3.2.2.3 Developing a CONVERGENCE Engine

Developing a CONVERGENCE Engine requires general knowledge of the CONVERGENCE/MXM architecture. The starting points are the convergence-core and the mxm-core projects. These specify the set of interfaces that each engine has to implement. For example, for the Digital Item TE, mxm-core specifies the following interface for the engine itself:

```
public interface DIEngine extends MXMEngine{
public DIEngineSchemaHandler getDIEngineSchemaHandler() throws
MXMEngineException;
...
}
```

mxm-core specifies additional interfaces for schema and technology handlers, for example:

```
public interface DIDL extends MXMCreatorAndParser,
org.iso.mpeg.mxm.tEngine.digitalitemTE.schemahandler.didl_mp21.DIDL{
public List<DIDLInfo> getMPEGMDIDLInfo() throws MXMEngineException;
public void setMPEGMDIDLInfo(List<DIDLInfo> infos) throws MXMException;
...
}
```

The current implementation of the Engines uses the mxm-dataobject module. This provides easy parsing and creation of XML structures, while remaining decoupled from the actual implementation of the engines. This makes it possible to use different XML parsers at the same time. The mxm-dataobject package contains:

- The JAXB's classes for the MPEG-M schemas (so-called level0 classes)
- A set of abstract utility classes facilitating the use of the JAXB representations (called level1 classes)

Each of these abstract utility classes extends a common super class (SuperAbstractLevel1). These classes are customized for each individual Engine, providing methods to create/parse/manage the xml representation of the Engine data. For example, the DIDL structure representing the Digital Item MPEG-M schema (that wraps the DIDL Type level0 class), can be written as follows:

```
public class SuperDidlAbstractLevel1 extends SuperAbstractLevel1 {

public SuperDidlAbstractLevel1(Object type, JAXBContext jc) throws
MXMEngineException {
super(type, jc);
}

public SuperDidlAbstractLevel1(Object type) throws MXMEngineException {
super(type);
}
...
}
```

With the help of the mxm-dataobject project, we can now implement the createFile method in the Digital Item Engine:

```
public class DIDL extends DIDL_mp21 implements
org.iso.mpeg.mxm.tEngine.digitalitemTE.schemahandler.didl.DIDL {

@Override
public void createFile(File f) throws MXMException {
(new SuperDidlAbstractLevel1(didltype)).createFile(f);
}
...
}
```

The DIDL object (level1 class) comes from the mxm-dataobject project and the createFile functionality comes from the SuperDidlAbstractLevel1 class, a customized version of the SuperAbstractLevel1 for the engine.

3.2.2.4 Using an MXM Engine API from an MXM application

A CONVERGENCE/MXM application can retrieve a generic CONVERGENCE/MXM Engine instance by specifying its name and properties in the MXM Configuration File and writing code such as the following:

```
File f = new
File(getClass().getClassLoader().getResource("MXMConfiguration.xml").getFil
e());
MXM mxm = MXM.getInstance(f);

//retrieve the engine
DIEngine die = (DIEngine)
mxm.getDefaultEngine(MXMEngineName.DigitalItemTE.toString());

//invoke a dummy API on the engine
DIParser dciParser = didEngine.getDIParser();
```

3.2.2.5 Extending an MXM Engine

CONVERGENCE/MXM Engine can be extended using normal JAVA programming techniques. In particular, developers can extend the convergence-core and mxm-core APIs to meet specific needs. Obviously, extended versions of Engines will be interoperable with other

CONVERGENCE/MXM only to the extent they comply with CONVERGENCE/MXM concepts.

To link CONVERGENCE/MXM Engines to application-specific engines, which are not part of the CONVERGENCE/MXM Engines, it is necessary to follow the steps defined below.

1) Define a class for the new Engine and its interface. The class should be an extension of MXMEngine.java, e.g.

```
public interface ConetEngine extends MXMEngine {
      public abstract RegistryParser getRegistryParser();
...
}
```

2) Define the name of the new Engine in an enumeration similar to MXMEngineName.java, e.g.

```
public enum MyEngineName {
MatchTE,
CDSTE,
ConetTE;
public String value() {return name();}
public static MyEngineName fromValue(String v) {return valueOf(v);}
}
```

3) Implement the interface for the new Engine, following the specifications for MXM interfaces e.g.

```
public class ConetTE extends MXMAbstractEngine implements ConetEngine
{
      ...
}
```

4) Create an application instance of the new engine in the same way as for any other MXM engine (the MXM configuration file can be written as usual) e.g.

```
File f = new
File(getClass().getClassLoader().getResource("MXMConfiguration.xml").getFil
e());
MXM mxm = MXM.getInstance(f);
//retrieve the engine
ConetTE oe = (ConetTE)
mxm.getDefaultEngine(MyEngineName.ConetTE.toString());
```

3.2.3 Implementation of Elementary Services

The elementary services provide a wrapped implementation of a protocol and one or more technology engines, managed by a dedicated *orchestrator engine*, exposed as a web service. Every time an elementary service is called, usually by the *remote* module of a protocol engine, the service parses the request, which is based on the standard protocol schemas provided in MPEG-M Part 4, makes the call to the corresponding orchestrator engine and returns the standard response. Elementary services can be seen as applications (the web service part), deployed in an application server, that use the CONVERGENCE/MXM middleware to perform their operations.

3.3 The MXM Configuration File

The MXM Configuration file is physically available in the mxm-core repository as *mxmconfiguration.xsd*. The schema provides an abstract representation of an object's characteristics and relationship to other objects. The XSD representation is written in XML. This means that it does not have to be processed by a parser.

The Figure below provides an overview of the MXM configuration schema.

Figure 1: Overview of the MXM Configuration schema

The schema comprises two main elements:

- MXMConfiguration – this contains the core parameters for the application and its location.
- MXMEngine – this specifies the type of each MXM engine and its relationships and dependencies with respect to other engines.

The *MXMConfiguration* element belongs to the MXM Configuration complex type, which contains two elements, the first describing the properties of the MXM object; the second containing references to the MXM Engines it supports.

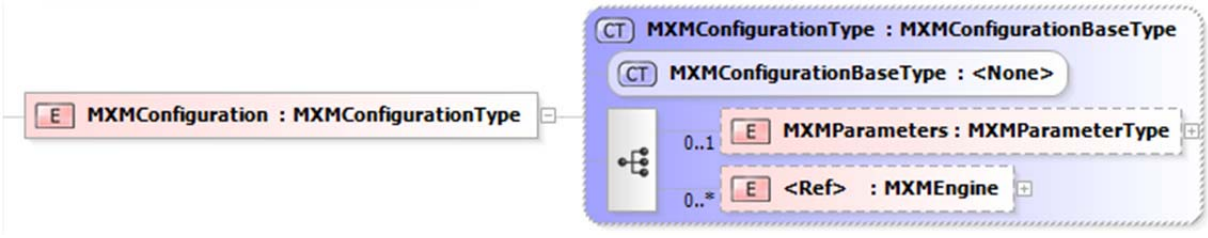

Figure 2: Structure of the MXMConfiguration element

MXMParameters is of type *MXMParameterType*, which specifies the use of a key-value pair as parameter, the use of complex parameters using a free representation as well as the location of the engine that the corresponding parameters are referring to.

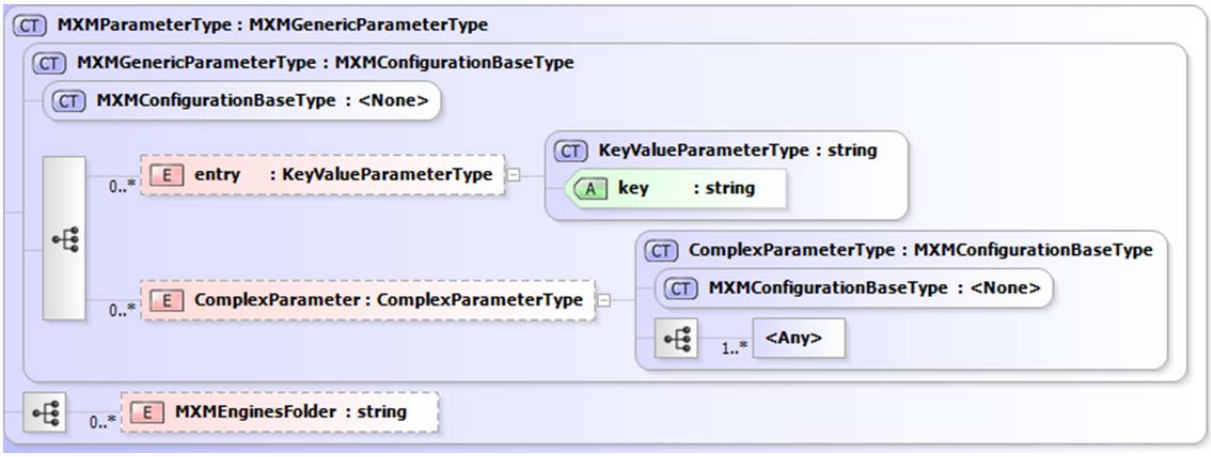

Figure 3: Structure of the MXMParameters complex type

The *MXMEngine* element belongs to the *MXMEngineType*. This complex type extends the properties of the *MXMConfigurationBaseType* and supports the following attributes:

- Id – identification of the MXM Engine
- Type – defines the type of the MXM Engine as listed in the corresponding MXMEngineName enumeration
- HasSchemaHandler – a Boolean attribute (true/false) that specifies if the engine has a schema handler module
- HasTechnologyHandler - a Boolean attribute(true/false) that specifies if the engine carries a technology handler
- IsDefaultEngine – a Boolean attribute (true/false) that indicates whether the given engine implementation can be used as the default one, in case different implementations are loaded.

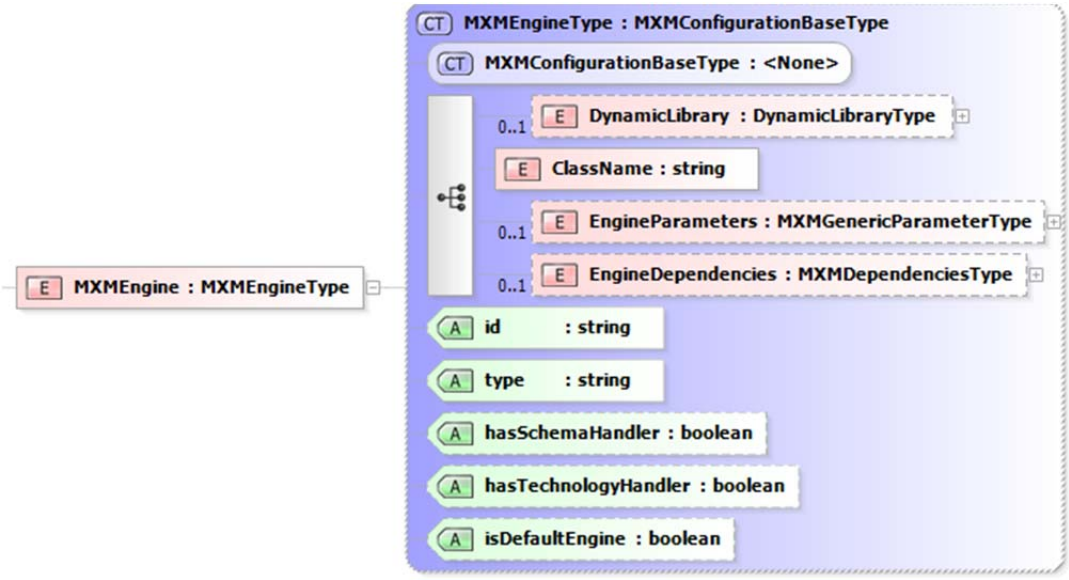

Figure 4: Structure of the MXMEngine element

Each engine uses a *MXMConfiguration.xml* file where specifies all the engines' attributes, elements and dependencies. Where necessary, the validity of this file can be checked against *mxmconfiguration.xsd*. This requires that it should be associated with the xsd file, as shown in the figure below.

```
<?xml version="1.0" encoding="UTF-8"?>
<mxm:MXMConfiguration xmlns: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 ">
```

Figure 5: Example of an association to the MXM Configuration file

4 Network Implementation

4.1 Implementation of Video Streaming Service Over CONET

This section briefly describes the implementation of a Video Streaming Service on top of CONET. As described in Section 4 of D5.2 [3] (and in [4]), the forwarding/routing plane of CONET is based on the CCNx engine and on the Lookup and Cache (L&C) infrastructure described in D5.1 [5].

4.2 Software Implementation for Individual Streaming

Figure 6 describes the implementation of the software used by the Client, the Server and the Publisher.

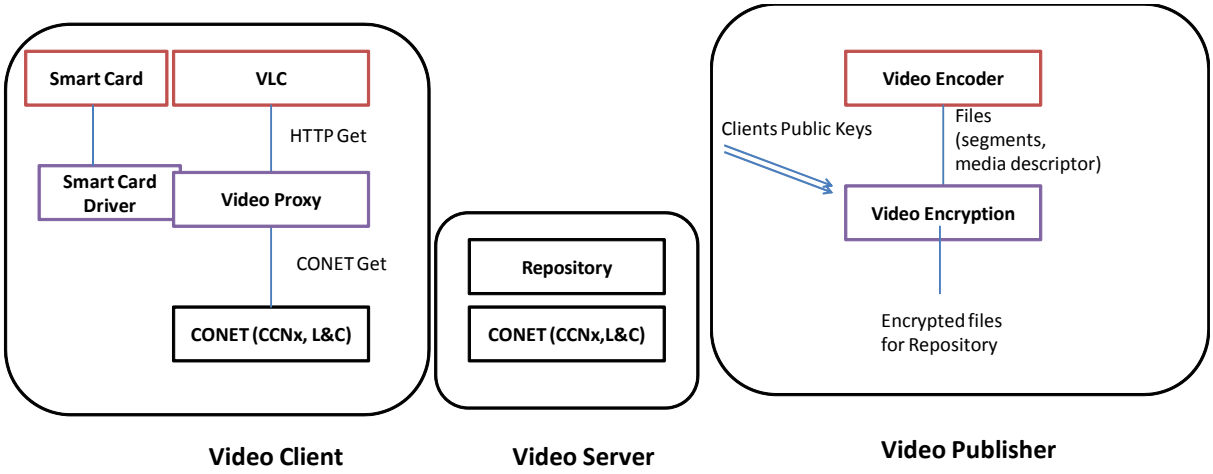

Figure 6: Software implementation for video streaming

4.2.1 Video Publisher

The Video Publisher uses an encoding tool that encodes the video in a format compatible with MPEG DASH [6] or with the Apple Live Streaming Format [7].

The output of the Encoder is a set of files, containing segments of the video streams. A segment is a part of the whole video, e.g. a sequence of 4 sec. Each file has a name e.g. "foo.eu/video#*x*.ts", where *#x* is the segment number.

In addition to video segments, the Encoder produces a media descriptor file that includes information on the names of the segments, the timing, the rate, etc.

If access to the video stream is restricted to a limited set of users, the Publisher encrypts the files produced by the Encoder. Encryption uses a simple but effective group encryption

technique based on traditional public key cryptography. In this scheme, the publisher selects a symmetric key (SK), encrypts the key SK with all the public keys of the enabled clients ($PK_{client}(SK)$) and encrypts the file with the symmetric key SK. The set of couples <clientId (e.g. the email of the client), encrypted key $PK_{client}(SK)$> and the encrypted file SK(file) are packaged in a single encrypted data-unit, as shown in Figure 7.

Obviously, this approach suffers a scalability problem when the recipient users are unknown a-priori or when their number is very large. To support these cases, the CONVERGENCE project is investigating approaches based on Attribute Base Cryptography.

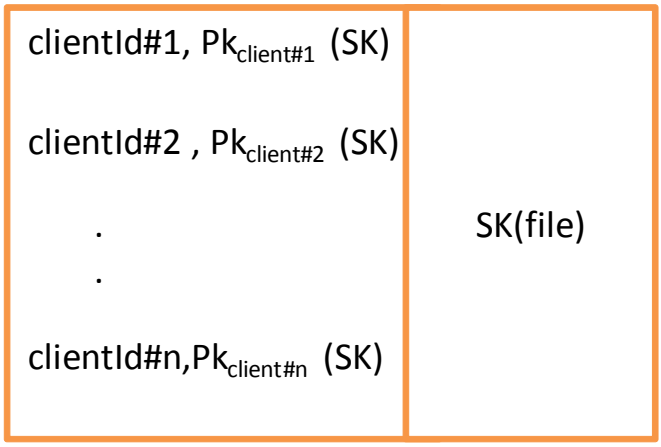

Figure 7: Public-key encryption of named-data

All the video files produced by the Encoder (encrypted or otherwise) are packaged as CONET named-data items, whose names are the file names. As discussed in D3.2 [8], CONET adopts the hierarchical naming format Principal/Label. For instance, in the name, “foo.eu/video1.ts”, the string “foo.eu” is the principal and “video1.ts” is the Label. After the production of the named-data items, the publisher delivers these named-data items to a video server, for distribution over CONET.

4.3 Video Server

The Video Server is a CONET node, which provides a Repository of named-data items, i.e. of video files. The video server stores the video files in the Repository and updates the Lookup-and-Cache routing plane to allow routing of requests for named-data items associated with the video files. Thus, if the Principal part of the Network Identifiers (NID) of the video files is “foo.eu”, the server will configure Lookup-and-Cache routing so that all requests for named-data items with this Principal are routed towards the Repository.

When a request for a video file, e.g. “foo.eu/video1.ts” reaches the Video Server, the Repository returns the requested named-data item.

4.4 Video Client

The video client device uses the Video Lan (aka, VLC) tool to play the video stream. Currently, VLC uses HTTP transport to fetch the video files, i.e. the media descriptor and the video segments. To interface VLC with CONET, we have implemented an HTTP/CONET video proxy, which:

1) Converts the HTTP GET operation into CONET GET operation (see D3.1 [9]).
2) Decrypts the received video file, if this is required.

Video decryption is based on a smart card. The smart card contains the private key of the user and is able to decrypt data items encrypted with the user’s public key.

When the proxy receives an encrypted named-data item, it singles out the couple <clientId, $PK_{client}(SK)$> using the clientId. It then passes the encrypted symmetric key $PK_{client}(SK)$ to the smart card, which decrypts it and returns the symmetric key SK. The proxy then uses SK to decode the file SK(file) enclosed in the named-data item and makes it available to the VLC client through a local HTTP connection.

4.5 Software Implementation of Cooperative Streaming for Cellular Network

As described in D5.2 [3], a node acts as a Wi-Fi local proxy for a set of video segments, reducing the number of times nearby wireless devices download the same segment over the expensive 3G interface.

A node advertises its role as a local proxy for a given segment, by broadcasting a “signalling” named-data item over a multicast Wi-Fi channel. This item contains the IP address of the node; the name of the item is “/prd/*segmentNID*”, where *segmentNID* is the network identifier of the video segment, e.g. “foo/video1.ts”.

When the VLC requests the download of a video segment, the requesting node begins by performing a CONET GET operation for “/prd/*segmentNID*”, allowing it to eventually discover the presence of a local proxy node. If it finds a local proxy, it inserts the entry <*segmentNID, localProxyIP*> in the CONET routing table, where localProxyIP is the IP address of the local proxy. It can now download the named-data *segmentNID* from the cache of the local proxy via WiFi, rather than via the expensive 3G interface. If it does not find a local proxy, it downloads the named-data “*segmentNID*” via 3G, it caches the segment, it publishes the named-data *segmentNID* and it becomes a local proxy itself for of the video *segmentNID*.

Every time a node downloads a segment from a local proxy, it sets a flag, ensuring that it will itself become the local proxy for the next video segment that does not have yet a proxy. This ensures that the cost of the 3G interface is shared fairly.

Figure 8 shows the software architecture of the video client. The main difference with respect to Figure 6, is the presence of the ProximityRIB module, which is a RAM based repository that stores the named-data items used by a node to advertise its role of local proxy for specific segments (i.e. all name-data items with name of type "/prd/*segmentNID*"). In addition, this module handles the cooperation logic described earlier.

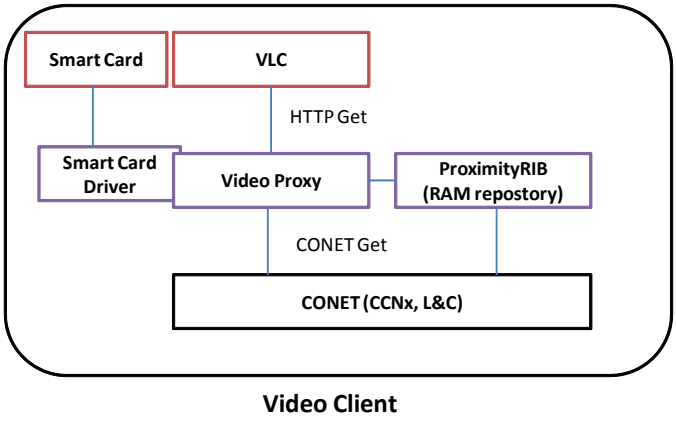

Figure 8: Software implementation of video client in case of cooperative streaming for Cellular Networks

5 References

[1] ISO/IEC CD 23006-3, Information Technology – Multimedia service platform technologies – Part 3: Reference Software and Conformance.

[2] ISO/IEC CD 23006-2, Information Technology – Multimedia service platform technologies – Part 2: MPEG extensible middleware (MXM) API.

[3] CONVERGENCE project deliverable D5.2 “Intermediate Protocol Architecture”, 2012

[4] A. Detti, M. Pomposini, N. Blefari-Melazzi, S. Salsano, A. Bragagnini, “Offloading cellular networks with Information-Centric Networking: the case of video streaming” IEEE International Symposium on a World of Wireless Mobile and Multimedia Networks (WoWMoM 2012), S. Francisco, California, June 25-28, 2012

[5] CONVERGENCE project deliverable D5.1 “Requirements and Initial Protocol Architecture”, 2011

[6] “Dynamic adaptive streaming over HTTP (DASH) -- Part 1: Media presentation description and segment formats”, ISO/IEC 23009-1:2012

[7] R. Pantos, “HTTP Live Streaming” Internet draft, http://tools.ietf.org/html/draft-pantos-http-live-streaming-08, March 2012, IETF.

[8] CONVERGENCE project deliverable D3.2 “System Architecture”, 2011

[9] CONVERGENCE project deliverable D3.1 “Definition of the system services and functional components”, 2010