THE ASSET ARCHITECTURE - INTEGRATING MEDIA APPLICATIONS AND PRODUCTS THROUGH A UNIFIED API

M.Cordeiro[1], P. Viana[1,2], J. Ruela[1], M. Body[3], B. Cousin[3], D. Bommart[4], G. Ferrari[5], M. Strambini[5], W. Bernet[6], E. Müller[6], M. Laurentin[7], B. Algayres[7], S. Daulard[8], I. Hoentsch[9], T. Marx[9]

[1]INESC Porto-Portugal, [2]ISEP-Portugal, [3]INRIA-France, [4]HP-France, [5]SHS-Italy, [6]A.N.N.-Germany, [7]FPDI-France, [8]THOMSON-France, ,[6]IRT-Gernamy

ABSTRACT

Applications and products currently available for the broadcasting market are vertically integrated or proprietary. They are based on components requiring specific and costly development to interoperate and do rely typically on a single manufacturer or system integrator. Hence they are not fully compliant with broadcasters' requirements.

ASSET is an European funded project whose main goal is to overcome the limitations of custom specific implementations of a digital system for TV content creation. These limitations are generally due to the misfit of interfaces between software layers, proprietary APIs of equipment from different vendors and lack of generalized middleware for multimedia content management with openly defined interfaces.

Besides presenting the ASSET proposed architecture and concepts, this paper describes the prototype under development to test and demonstrate the project proposals.

INTRODUCTION

The main goal of the ASSET project (IST-2001-37379 - Architectural Solutions for Services Enhancing digital Television) [1][2] is to overcome the limitations for custom specific implementation of a complete digital system for TV content creation. These limitations are generally due to lack of connectivity and interoperability between equipment and between applications, the lack of generalised middleware for multimedia content management with openly defined interfaces and the misfit of interfaces between software layers or between the proprietary APIs of the equipment.

Technical solutions available on the market are vertically integrated or proprietary. They are based on components requiring specific and costly development to interoperate and do rely typically on a single manufacturer or system integrator. Therefore they are not compliant with requirements of end-users (in particular, the broadcasters).

These limitations are usually due to:

- Databases of different type with respect to access and data mode
- Lack of a standard for multimedia Content exchange
- Proprietary APIs and protocols to control and interconnect components and applications

The ASSET project is defining and developing a software architecture and the corresponding

technologies necessary for a unified management of digital TV content which covers the complete operational workflow, including acquisition, creation, editing, control, storage, broadcasting, publishing and archiving of digital TV content.

ASSET is therefore defining and/or implementing:

- A harmonised data model and interfaces for accessing and manipulating the multimedia objects covering the essence (audio and video) and the associated metadata, based on existing standardisation work.
- A Command and Control System between the different ASSET components. ASSET is defining a middleware layer, based on distributed technology, in order to expose an interface for controlling the different ASSET devices and servers.
- A reliable way to interchange audio-visual program material, system and descriptive metadata between the different ASSET components as integral part of a production and content management system.

The partners of the ASSET project (HP, THOMSON, Dalet a.n.n, INESC Porto, INRIA, IRT, FPDI, SHS Multimedia) are exploiting open standards and emerging concepts and technologies like MXF [4], standard data models for describing essence, XML [3] and distributed systems technologies, for defining the concept of an Asset Middleware. Demonstration of the approach is based on the development of a simple workflow of a Newsroom Platform prototype based on the ASSET architecture and API.

ASSET ARCHITECTURE

The ASSET architecture is based upon a software framework – *the ASSET framework* – composed by a set of three standard interfaces and protocols for applications and products working together in an integrated environment. Applications communicate with the framework using the *ASSET Public API*, which allows them:

- to communicate with any other application,
- to access the public services provided by the framework,
- to use additional functionalities implemented by third party integrators as aggregated services.

Products from different manufacturers are integrated in the ASSET framework either natively (using an ASSET agent) or via an ASSET proxy. They are controlled and managed through the *Media ASSET Bus API*, based on XML service schema definitions. This API ensures an easy and seamless integration of devices and products in the framework.

An *ASSET Private API* is defined to access core services of the framework (common services) that enable the workability of this integrated environment. This API is only used internally in order to guarantee the overall system integrity.

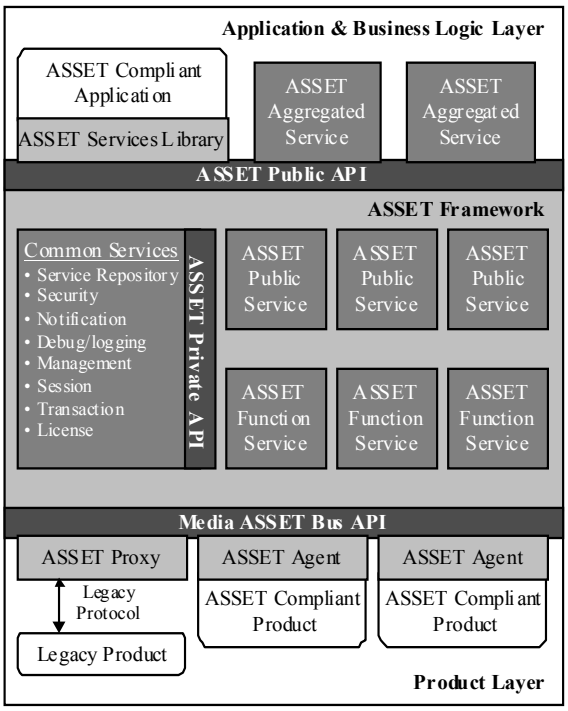

Figure 1- ASSET Architecture

Figure 1 illustrates the different components of

the ASSET architecture.

ASSET Components

The ASSET Architecture defines a number of concepts, components and functions that enable the implementation of an ASSET Compliant Framework. The core of the ASSET Framework consists of three main components:

- The *ASSET Public Services* expose the mandatory services of the ASSET framework to the applications and to the aggregated services through the ASSET Public API. These services provide a minimum set of multimedia functionality, sufficiently rich and extensible to not limit the system efficiency. They ensure the consistency and integrity of the system by handling the internal logic (e.g. access right, resource allocation, etc.) required for each operation.
- The *ASSET Common Services* provide implementation of key infrastructure requirements such as security, logging, notification, resource management, etc. This allows a uniform and single implementation of these services throughout the framework. They expose a private API that can be only used by ASSET Public Services.
- The *ASSET Function Services* provide an abstraction of functionalities (encoder, recorder, player, etc.) to the ASSET public services. They hide the specificities of the different interconnected products (e.g., for a public service, a VTR output and a Video Server output are considered as two system-wide logical output ports).

At the Application and Business Logic layer, three components are introduced:

- The *ASSET Compliant Applications* are the top level ASSET software components. They use the ASSET Public API to access services provided by the ASSET Framework and, optionally, by ASSET aggregated services.
- The *ASSET Services Library* is a software component included in (or linked with) an application, which makes it compliant with the ASSET framework and gives it access to the ASSET public services.
- The *ASSET Aggregated Service*s implement additional business logic on top of public services (or even other aggregated services). They register in the framework as new services available for ASSET compliant applications (e.g., complex workflows may be specified as aggregated services and then be made available to other connected applications or aggregated services).

At the product layer, we call *product* a manageable hardware or software component that implements one or several common functions (e.g., most of the Video Server products implement a Recorder function, a Player function and a Storage function) and logical components (logical ports, repositories, etc.). Two ways exist to make a product compatible with the ASSET framework:

- An *ASSET Compliant Product* is a product that is managed by the framework through a built-in *ASSET Agent*.
- A *Legacy Product* is a product that has not (or cannot have) a built-in ASSET agent. The ASSET framework can nonetheless manage such a product through an external software module called an *ASSET Proxy* (e.g., a VTR is not capable of including a built-in ASSET agent and has to be connected through an ASSET Proxy).

Media ASSET Bus

Most of the ASSET framework is built on top of a software bus called the Media ASSET Bus or MAB.

The goal of the MAB is to provide support for the integration of the widest range of products within the ASSET framework, independently of the underlying operating system environment and protocols.

The MAB defines a set of standard interfaces and synchronization processes, which ensure a seamless interoperability between ASSET components and products. These interfaces allow any third party media application or product to be integrated with the Media ASSET Bus by developing a simple software adapter in an agent or a proxy. A MAB Software Development Kit (SDK) is provided in order to facilitate this task. It gives a uniform way of connecting devices, registering services and exchanging messages within the ASSET framework.

The integrated environment offered by the MAB is based on a Transport Abstraction Layer (TAL) that takes care of message exchanges. Messages have XML format, thus ensuring flexibility and adaptability of the ASSET solution.

The concept of the Media ASSET Bus is illustrated in Figure 2. Different ASSET components and products are interconnected via software adapters on top of the MAB SDK.

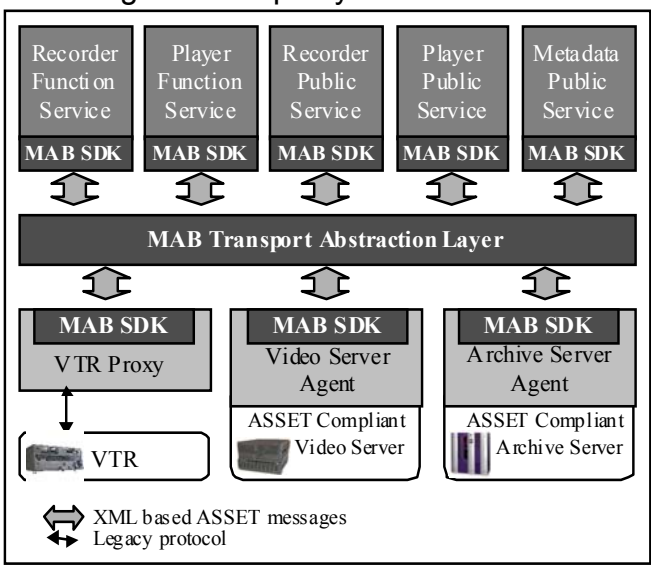

Figure 2 – Media ASSET Bus

DEMONSTRATION PLATFORM

The demonstration platform, which is being implemented as part of the ASSET project, has the purpose of showing the feasibility and simplicity of integrating different broadcast systems via the ASSET middleware. It should also serve as a reference system, where end users, administrators and integrators can test and verify the implemented functionality against the requirements, thus validating the result of the project.

The demonstrator uses one client and two providers of different manufacturers to emulate a simple workflow of a News Platform. The client is a newsroom application that controls the ingest process and lists the content stored on the online video server. The providers are an ingest server and an online storage system provided by a video server. Both the providers and the client application are connected to the ASSET middleware. All the Command & Control operations and the data exchange are controlled by the components of the ASSET middleware.

Interaction between the different components used in the demonstrator is presented in Figure 3.

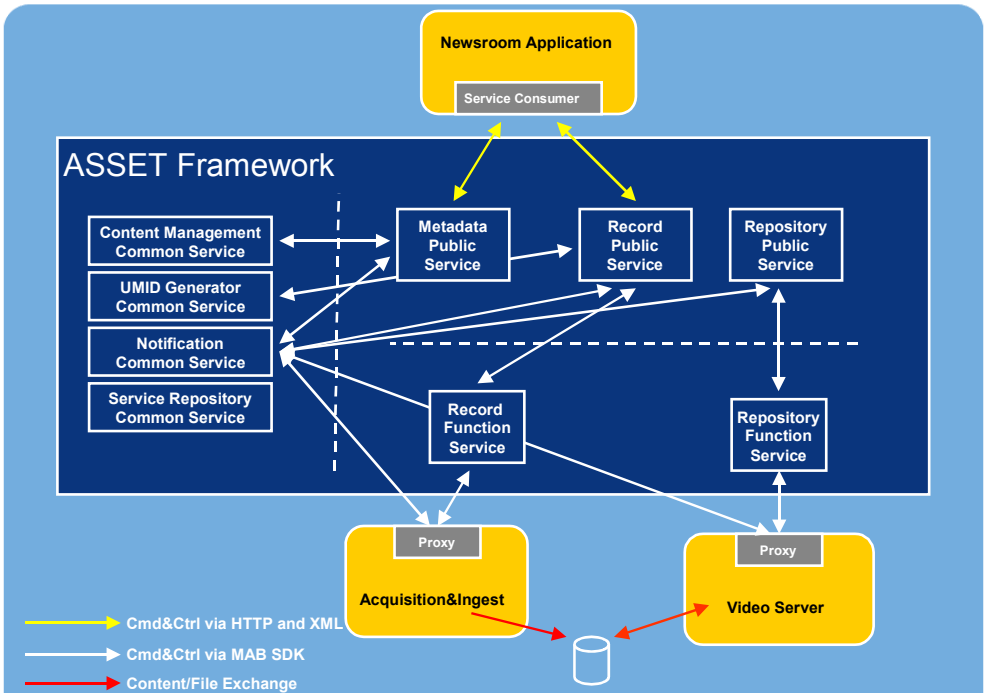

Figure 3 – ASSET demonstration platform components

Demonstration Platform Components

The demonstration platform will make the following set of Public, Common and Function services available:

1. Public Services
    a. Record PS: provides an interface (API) to start, stop and restart the recording of the pre-programmed feeds.
    b. Metadata PS: is the central access point for accessing, creating and modifying any kind of metadata in the system (content management metadata, descriptive metadata, structural metadata).
    c. Repository PS: provides the interface for the applications to access and to modify content stored in any repository (online, near line).
1. Common Services
    a. Service Repository CS: provides a mechanism for services (Common and Function services) to register to the framework so that they can be addressed and accessed by other services or applications.
    b. Notification CS: provides a subscription mechanism for other services and applications to be notified upon events like creation and modifications to objects.
    c. UMID Generator CS: provides a unique material identifier according to the SMPTE standard [5].
    d. Content Management Metadata CS: maintains the link between a UMID and the repositories where the material is located.
1. Function Services
    a. Record FS: provides an abstraction layer (entities, data exchanged) for

controlling Audio/Video input ports (e.g. sending differed commands to control the recording of a media asset). In the demonstrator scenario the Record public service uses the Record function to control the input port of the acquisition server.

b. Repository FS: provides an abstraction layer (entities, data exchanged) for controlling all the storage repositories of a system. The repository for the demonstrator is an online storage video server storing the material ingested by the Acquisition service.

Demonstration Hardware Architecture

The demonstrator components are connected using different network technologies as presented in Figure 4.

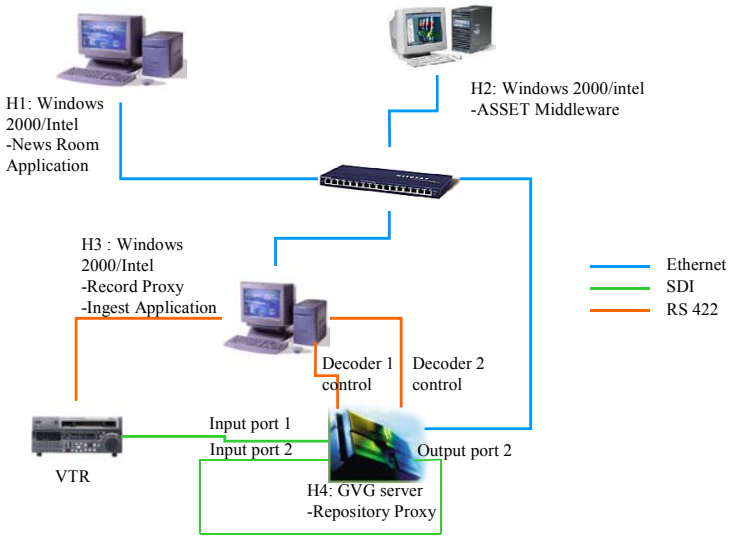

Figure 4 – ASSET network architecture for the demonstrator platform

The Newsroom application displays the status of the ingest operation and lists all content stored on the online video server. Incoming material (A/V-feeds) are digitised with different technical qualities. High-resolution copies are recorded on the video server. Each time a recording is initiated, a new and unique UMID will be associated to the new feed. The video server informs (notifies) the ASSET framework about the new content creation and status changes.

Figure 4 presents the network architecture of the demonstrator platform.

Register Record and Repository Functions - Use Case

Some use cases were developed to model the behaviour and the interaction between different modules in the demonstrator. Figure 5 presents the Record and Repository FS registration into the ASSET framework. The following steps are executed:

A0 – All the PS subscribe for notifications to the Notification CS in order to receive notifications they are interested in – the Metadata PS to receive all modifications regarding the creation, deletion, move, and modification of content, and the Record and Repository PS to receive notifications about the corresponding newly registered functions.

A1 – The Record and Repository Proxies are registering to the ASSET framework. The Service Repository CS creates the corresponding function services acting as client stubs for the proxies. After that, the corresponding PS are notified about the new registrations through the notification common service. In the general case, the public services would read the pre-defined configuration from the Configuration Management CS. In the demonstrator scenario the configuration for the proxies are stored in configuration files read by the corresponding PS. The PS initialises the corresponding proxy by calling the function service' `Initialise` command passing the configuration in the XML data parameter.

A2 – After its registration, the Repository proxy sends a notification describing its content. This notification is received by the Metadata PS, which updates the Content Management CS information regarding the content location (repository) and media assets structures.

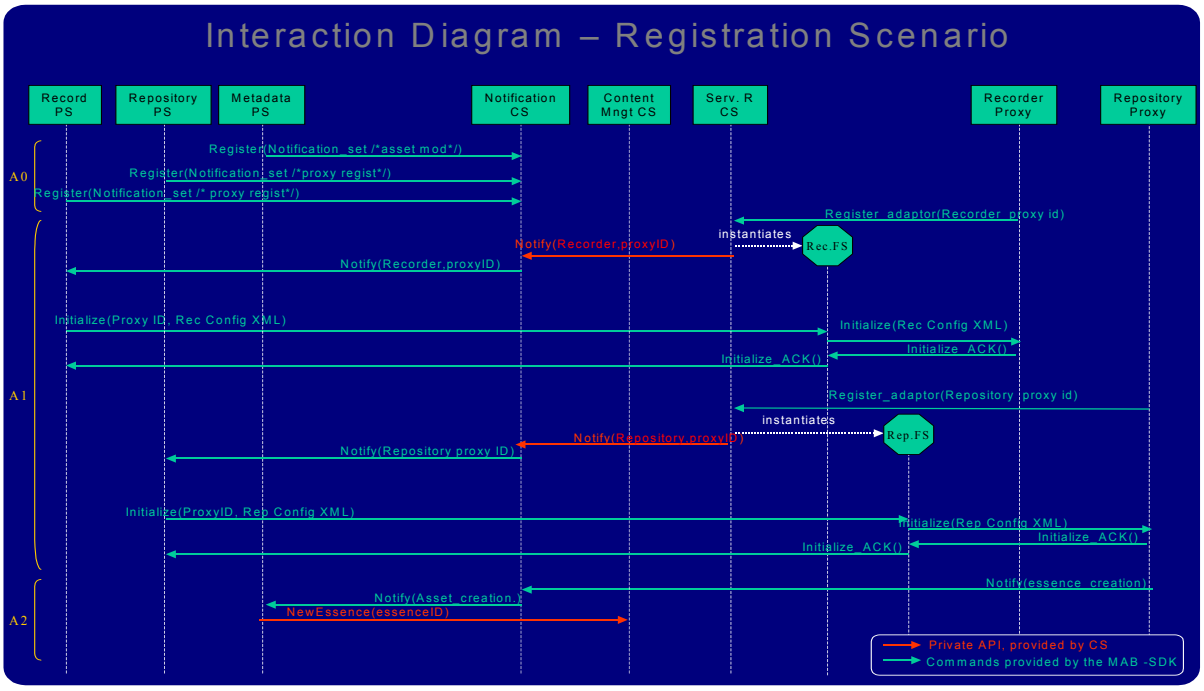

Figure 5 – ASSET demonstration Registration Scenario

CONCLUSIONS

This paper gives an overview of the work developed within the scope of the ASSET project. The software architecture and the main ASSET concepts are described and a simplified prototype that is expected to test and validate the project approach is presented.

This work is expected to improve the interconnection between equipment and media applications in digital TV environments, solving some of the problems users and system integrators face currently.

REFERENCES

[1] ASSET web site – http://www.ist-asset.com

[2] Paula Viana et all, 2003. A Unified Solution for the Integration of Media Applications and Products in Broadcaster Environments – The ASSET Architecture. Proceedings of the NAB Conference

[3] W3C, 2000. Extensible Markup Language (XML) 1.0 (Second Edition), W3C Recommendation. World Wide Consortium, http://www.w3c.org/TR/REC-xml, October 6.

[4] SMPTE, 2003. Material Exchange Format (MXF); File Format Specification, SMPTE Proposed Standard 377M

[5] SMPTE, 2000. Unique Material Identifier (UMID), SMPTE Standard 330M-2000