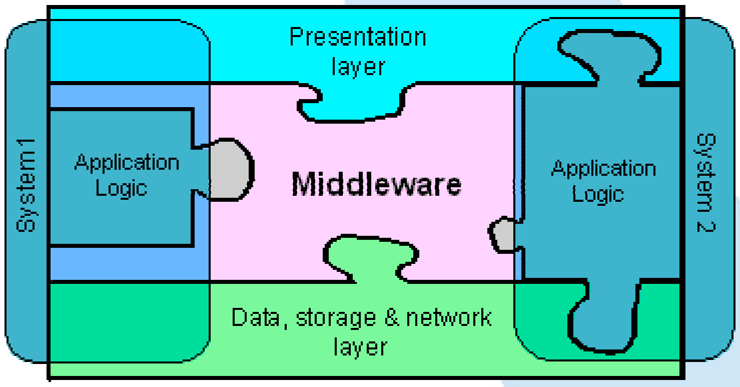
EBU – TECH 3300s

EBU·UER

Supplement to the Middleware Report

System integration in broadcast environments



Geneva
February 2005

Foreword

This document is the final report of the EBU project group on Middleware in Distributed programme Production (P/MDP).

It is intended to be of use both to IT people working in broadcasting and to broadcasting people working with IT, and to both managerial and technical in either field.

Some parts of it will seem over-simplified and some parts over-complex, but they will be different parts for different people. This is the inevitable consequence of trying to produce such a broadly accessible document.

Further, it will be seen differently depending on how far along the path of 'system integration' that the reader's organisation has progressed. It might be a welcome source of information to supplement an existing development or it might be an alarming introduction to what the future holds.

The EBU P/MDP Project Group was established in the autumn of 2003 to study the topic of Middleware, as the term 'Middleware' is used extensively in technical discussions but often lacks a clear definition.

Triggered by this, the group started out to define Middleware terminology as well as to produce an in-depth study of the problems and the opportunities. In doing so, it became clear that what the group was really studying was 'System Integration Architectures'.

The group subsequently gathered Members' experiences and held meetings with manufacturers' representatives to discuss this topic[1]. Requirements for Middleware were identified and recommendations for future work were defined, as were important elements of working with System Integration and also some 'Golden Rules'.

This document is the result of their work.

Reading this document

The document is split into two parts.

1. Tech 3300
   the main report is intended for a general readership. At the very least, read the Executive Summary.
2. **Tech 3300s (supplement)**
   this part is intended for specialists, R & D departments, and managers with special interests.

Both parts contain glossaries; as terms and definitions are very important in this work.

Please note that the chapter numbering in the supplement follows on from that in the main report.

Level of detail

The reader should be aware of the way that this work uses illustrations and metaphors. It is working at the edge of what everyday language can describe. Therefore, in trying to communicate the implications of some very complex structures and information as simply as possible, it has to use some terms that, to a specialist, might be seen as imprecise or inaccurate.

The reason for this is to try and make accessible some very important concepts - to describe that of which broadcasters should be aware, and what they should do, when working with System Integration Architectures.

Some of the more detailed definitions and descriptions that have still to be defined are suggested as future work.

[1] The P/MDP Group wishes to thank all participating vendors for their valuable contributions.

Table of contents

2. System integration architectures

2.1 Introduction and basic concepts

2.1.1 What is 'integration'?

In general, *integration* is a process that organisations go through when there is the need to make systems work together in an orchestrated way. So *integration* is not a rare occurrence, in fact it happens commonly in day-to-day operations. A simple example of a very common type of integration is when a new computer is plugged into an existing network environment. Another example could be the expansion of an existing file archive by adding new network attached file systems.

Of course not all cases of *integration* are so simple. Indeed, in a typical integration process a great deal of effort is spent on adapting systems' interfaces, understanding and designing the points of interactions between systems' components, and adding all of the intermediate pieces that make the whole system work.

What are the means by which these processes are possible? What are their features and characteristics? The answers to these questions can be found, starting with understanding what an *integration architecture* is.

2.1.2 System integration architecture

By *system integration architecture* we mean the set of strategies and methods that are used to solve system integration problems.

The architecture with which to achieve system integration may depend on various aspects, e.g. on your particular business model (products and operations), your particular IT-environment, which may or not be distributed around your organisation, and many others.

Nevertheless, it is possible to describe some common basic integration patterns and reusable components, to recommend tools, and to give guidance.

At a later stage, it might even be possible to set up some standard requirements for the vendors of the systems we are buying, encouraging them to provide an open and non-proprietary environment. This would simplify the future integration activities that can always be expected in the mid/long term of the life of any complex system.

Because the following description is relatively general, the integration architecture concepts presented here apply to planning, (post-)production, and play-out systems, as well as to administrative back-office and ERP systems. Attention to the real-time aspects of broadcasting is maintained throughout.

2.1.3 System analysis: an object-oriented approach

Systems, by their very nature, integrate a multiplicity of components. Component views provide the means to represent macro-functions or operations of a system. These views are extendable, in the sense that what can be seen as a system in one view, may appear as components (or subsystems) of a larger system in a more high-level view. Systems may thus act as subsystems in a larger context.

Object-oriented technology provides an intuitive way of modelling complex solutions by splitting them into convenient blocks, which themselves can be subdivided into further blocks. Each block (object) has attributes and can execute some procedures.

The entire value chain of a broadcast environment can be seen as being realised by a set of procedures belonging to a set of objects. The object-oriented approach requires that suppliers of procedures make their interfaces public. Also protocols for procedure requests from external entities should be made public.

One of the benefits of the object-oriented approach is that when the internals of an object are changed, usually little or no other objects need to be changed, because the changed object's interfaces remain the same. In addition, this lends itself to a wide re-use of existing software by allowing a new object to inherit all the capabilities of the previous object, building enhancements on top of these.

As the migration towards the object-oriented scenario progresses, new equipment is expected to work directly in the object domain. This more modular approach calls for new modelling tools to describe and develop functions, data structures, systems and their relationships. Also standards may evolve from the functions identified in current equipment to meet the requirements of new system elements as they are developed.

2.1.4 A classification of system structures

It is quite commonplace among system integration specialists to classify systems depending on their structural features. Typical classifications are: openness, layering, orientation (horizontal/vertical) and other similar concepts. Giving a complete and coherent characterisation of these dimensions is a complex task, and unfortunately beyond the scope of this report. However, some basic concepts can be introduced and explained, in order to clarify the terminology that is used:

- A system can be described as a set of organised components tailored to a particular use.
- The properties of a component include:
  - unique identification
  - operations
  - I/O parameters
  - behaviour
  - semantics
- Subsystems are sets of components with a specific purpose within the context of the overall system's purpose.
- A system has several classes of users. From the system's perspective external systems are seen as users as well.
- Each class of users has access to the system by means of a subset of its components.
- *Openness* is the possibility for a user, or an external system, to perform operations on system components.
- *Layering* refers to structuring systems in such a way that deeper components or subsystems can be accessed through higher level ones.

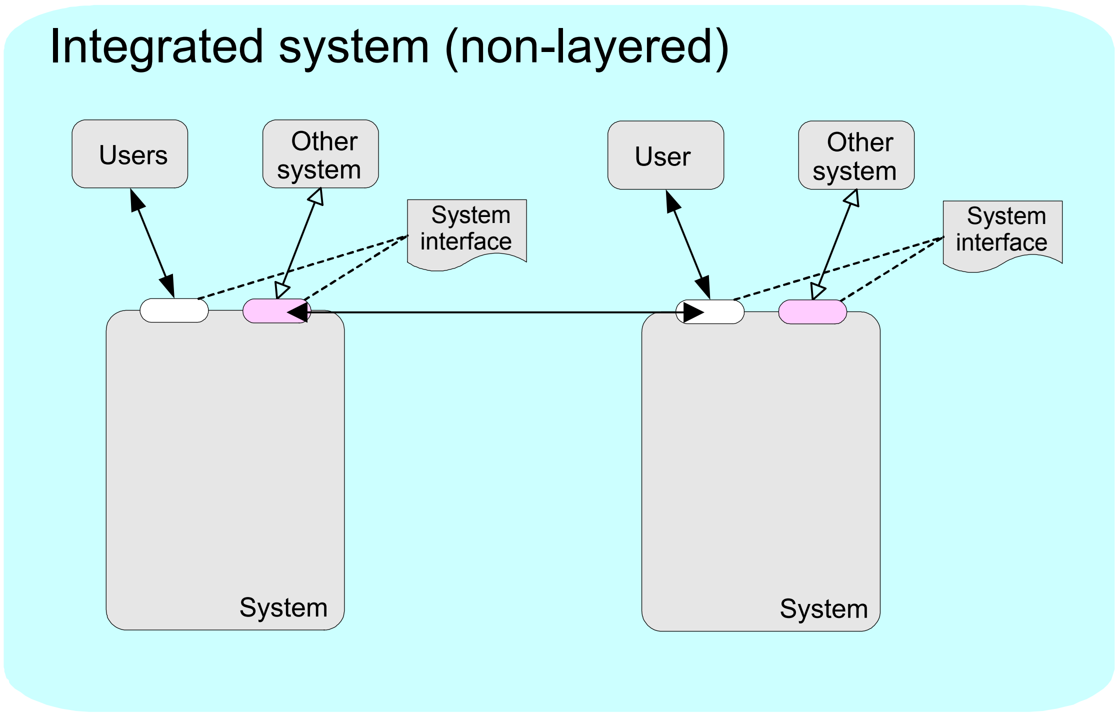


Figure 2.1 A non-layered integrated system

2.1.5 Why openness and layering are important

Systems that are structured as *open* and *layered* are interesting from the integration perspective, because the process of creating new systems is simplified. Namely:

- An *open* system may provide several possible points of integration.
- A *layered* system permits to limit the integration process to integrating higher-level layers only.

Open and layered systems offer the greatest opportunity for an easy integration process. This is why these two properties are so desirable for system structures.

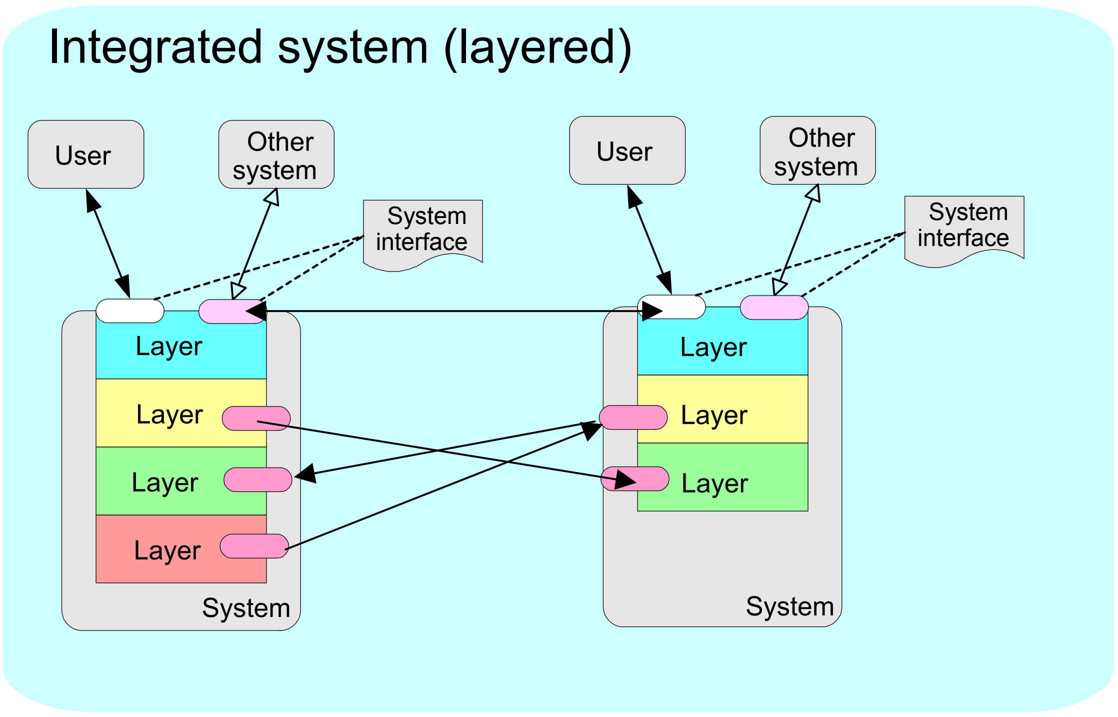


Figure 2.2 A layered integrated system

The roles that layers have reflect the component hierarchy to system users. Layers can include both hardware and software components with their specific interfaces. For the systems in a broadcast environment, in most cases a four-layer representation is suitable (Figure 2.3). In this chapter the four-layer model is used as a visual reference, but the concepts explained apply to other layered arrangements as well.

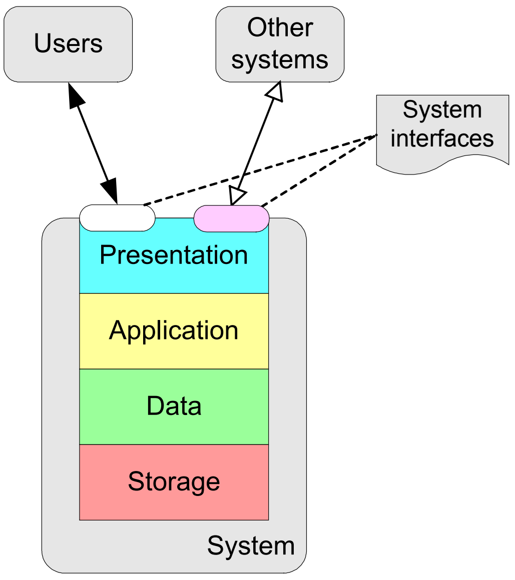


Figure 2.3 4-Layer system

Presentation layer: contains the set of components which provide system control and input/output for the system. The external environment interacts with the system via the presentation layer. Some components in this layer implement interactions with users via a human-oriented interface, other components interact with other systems via protocols and interfaces. The term ‘System Portal” is often used to denote this layer.

Application layer: contains the components which implement the "brain" of the system. It is the execution mechanism of the system, controlled by system users through the presentation layer.

Data layer: contains the set of components which provide access to data for input and output. This is generally a persistent system memory realised in one or several databases.

Storage layer: contains the set of components which provide physical storage of data in fileservers, tape libraries, etc. Usually all objects of this layer are managed through a Hierarchical Storage Management system.

At the layer's boundaries components can give access to their procedures and functions to any other component of the neighbour layer and vice versa.

2.2 Integration in the broadcast environment

Progress within the IT industry is changing the landscape of broadcast system development and exploitation. We can consider a broadcast system as a special type of information system. A description of a modern system could be:

- All inputs and outputs of a broadcast system can either be in the form of digital bitstreams or a physically transported storage medium;
- All parts of “persisted” content include essence and Metadata stored in file form;
- Digital data transportation is based on IT-standard facilities;
- User interfaces are generally located on a PC with specialised software;
- Most of the application processes are executed in the heart of system, (possibly on servers) and shared between many users;
- A user can connect to the system not only via local area network, but also as a remote WAN client.

But broadcast systems also have some particular characteristics:

- Large amounts of complex data;
- Many simultaneous users;
- Typically complex user interfaces;
- Complex business logic.
- Many points of interaction with other systems.

Another issue concerns the system's inability to change its functionality when it is in operation. Developing broadcast systems that support evolution in a seamless way is not an easy task; at an early design stage, some concepts may be confused or some potential functionality not foreseen, resulting in bad separations of concerns.

We believe that these 'problems' may be solved by a proper system integration strategy. Let’s try to break down the problem into two aspects:

- Heterogeneity
- Distribution

2.3 Heterogeneity

The first aspect (heterogeneity) consists of the variety of information domains and purposes that systems use and cover. Systems can be perfectly concentrated in a single location, but still be unable to communicate due to incompatible semantics, different data models or different behaviours.

In general, heterogeneity is the first problem to be faced, because this requires some typical and crucial processes of integration:

- Adaptation of parameters
- Adaptation of operation paradigm (asynchronous to synchronous and vice-versa)
- Adaptation of semantics
- Adaptation of behaviour (states and sequences of actions)

If these aspects are not catered for, then integration cannot be achieved at all. Unfortunately, there are no “off-the-shelf” solutions able to solve this class of problem systematically and automatically. On the other hand, very good examples of supporting tools and facilities can be found in many related areas, such as data mining, data warehousing, data discovering, workflow orchestration, transaction management, etc.

It is quite common in the broadcast environment to struggle with heterogeneous deployments, mainly due to the different times at which systems design took place, different responsibilities, different management control over deployed systems and different competence areas dealing with the same part of the value chain but with different views on it.

The most important technical strategies to face the problem of integrating heterogeneous systems are:

- **Message based integration** (copying data between systems)
  The same data may be needed in different systems and an automated exchange mechanism can provide copies of data in these systems and ensure data-integrity while carrying out this process. This can be set up as continuous or event-based structures, while the functionality could be provided by integration brokers.
- **Data-carrying integration** (Different systems access to the same data)
  Data is only located in one copy in one place. All systems are working real-time on the same instance of data. Integration is made by agreement on which data model is used between the systems, as well as the common logic used for reading and updating data. This is provided by a data-carrying integration platform, consisting primarily of an application server and a database.
- **Portal integration** (Wrapping of user interfaces to the systems)
  Used where users require a more integrated user dialog or user interface than is provided by the separate applications. Where individual customising of the user interface is required, where certain functions should be automated, or where single sign-on is needed, an integration of the user interface is needed. Web-based user interfaces provide integration by a portal.
- **Workflow** (Mechanisms to co-ordinate events in systems)
  Automating or semi-automating workflows and the use of functions or data integration. To support tightly connected or integrated business processes it should be possible in advance to define which actions take place when certain data are exchanged or updated. Workflow mechanisms can be found in each of the three strategies described above.

2.3.1 The Problem of “interoperability”

The strategies mentioned above assume implicitly that any relevant data models are shared and understood by the systems exchanging data, or at least that a data integration architecture has been previously agreed on. For example:

- For real-time broadband interaction, we have stable and widely adopted SDI/AES interfaces, where interoperability is no problem, as standards are extremely well developed.
- In the case of digital videotape recording, based on a few selected compression standards and physical media, there is also no problem with interoperability. That is because only a few well-known vendors can develop and produce these kinds of very complex hardware devices.

For products realised in software, broadcasters face having to deal with many different specifications. Many vendors either have not spent sufficient resources for using existing protocol specifications, or they have made a deliberate decision not to do so.

The result is the introduction of proprietary solutions. This forces broadcasters to have to deal with the problems of software component interoperability if flexible and open systems are to be achieved. This is a major problem when setting up an open System Integration Architecture.

2.3.2 Open and closed systems

One of the dangers experienced in a real-world system implementation follows the delivery of a number of “subsystems” or “system components” if the integrator cannot completely fulfil the broadcaster’s requirements. Of course this is only discovered when the system integration is completed.

Often a lack of clarity at an early design stage is the cause. E.g. some design concepts or process functionality may be confused or some potential functionality not foreseen. For any system that is specified from new, or existing systems being developed further, the overall process functionality has to be well understood.

Added to this, at any stage of system evolution, a broadcaster may need to add new requirements. This may involve altering the system functionality, requiring access to new components.

To help solve this problem, systems should be accessible not only at presentation (top-level) layers, but at any of the layers. This provides more ways to enhance the system’s functionality. Detailed modelling of the required processes at an early stage should help understand these needs (see chapter 4). Of course, functional access to different system layers could be severely limited if these are not open to developers other than the original manufacturer.

Figure 2.4 shows some examples of open systems, which are opposite to the fully closed system in Figure 2.1:

- Example a): a system open at the Application layer. Users of other systems can access the Application layer through their own Presentation interfaces, which can be developed without changes to the Presentation layer of the system shown.
- Example b): the system is now also open at the Data layer. Other systems can operate with records in system b)'s database.
- Example c): a fully open system. External access to the Storage layer is possible. This example reflects the separation of systems into components in functional layers.

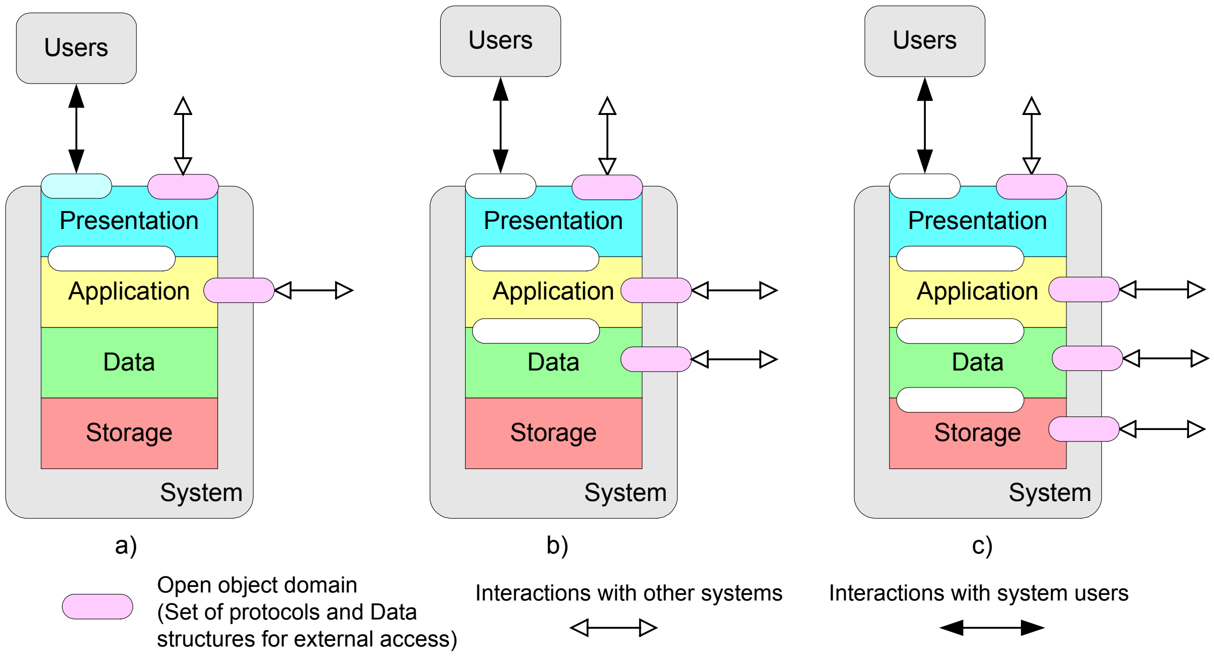

Users
Presentation
Application
Data
Storage
System
a)
b)
c)
Open object domain (Set of protocols and Data structures for external access)
Interactions with other systems
Interactions with system users

Figure 2.4 Types of open systems

Users should ask what degree of system openness is desirable for their circumstances. Fully documented protocols are desirable in any case. Note that the level of openness is important, as an open system at all layers with very limited exposure of its internal components still may not be able to interact with your other equipment in the way you might wish. You need a guarantee of system integrity and co-ordinated shared access to resources for internal and external requesters.

2.4 Distribution

A complex system can run on a single PC, but would more likely be distributed across many servers platforms and client PCs, all connected via local or wide area networks. Some examples:

- A big system has several nodes, which maintain specific workflow stages – Ingest, Production, Play Out, Archive, Administration etc. The overall workflow can be maintained only when the nodes (subsystems) are integrated.
- A big system (broadcast network) consists of clone subsystems (local stations), which maintain similar workflows. The purpose of integration is content exchange, facilitating the businesses of each station.

Let us assume that the components of a system are distributed via a LAN and users connect with the presentation components via TCP/IP, and let the storage facilities be concentrated in a SAN. In this case we would say the system layers are distributed over the network.

A system can be built as a very good single system, but isolated from another one. The interfacing then would be via external users and systems. We then say the two systems are co-existing. The main task of system integration is to turn co-existing systems into co-operating systems.

The simplest solution may seem to be to turn the systems into a single concentrated solution. But the products (e.g. programmes) we create are now more diverse and spread over more media platforms than they used to be. Also workflows are changing rapidly and the operations are spread out over different physical locations. At the same time, the technology used is also much more diverse and distributed.

In this situation, future changes may imply huge modifications in the system's structure and component interfacing. This means that any attempt to join systems in concentrated solutions is like an attempt to join islands into a continent. A better approach would be to consider another way of integration – namely to join systems as if they were an archipelago, consisting of IT-islands connected via software interfaces based on information highways, bridges and tunnels.

These can be seen as special purpose components designed to allow flexible and easy integrate-ability and interoperability among distributed systems. The assumption made is that the integration has been already been proved to be achievable at the semantic level (i.e. what, when and why to interact). For the "how", two technology frameworks have been successful in the market:

- **MOM -** Message Oriented Middleware

  Message-Oriented Middleware (MOM) are components providing the abstraction of a message queue that can be accessed across a network. It is a generalisation of the well-known operating system construct: the mailbox. It is very flexible in how it can be configured with the topology of programs that deposit and withdraw messages from a given queue.

  A message broker is an application-to-application middleware, composed of one or more software facilities, and capable of one-to-many, many-to-one and many-to-many message distribution, using queuing, prioritising, confirming and other transaction handshakes. Other logical rules for the transactions might be added.

- **SOA -** Service Oriented Architecture

  A service-oriented architecture (SOA) defines how two computing entities interact in such a way as to enable one entity to perform a unit of work on behalf of another. The unit of work is referred to as a service, and the service interactions are defined using a description language. Each interaction is self-contained and loosely coupled, so that each interaction is independent of any other.

  Simple Object Access Protocol (SOAP) -based Web services are becoming the most common implementation of SOA. However, there are non-Web-services implementations of SOA that provide similar benefits. The protocol independence of SOA means that different consumers can use services by communicating with the service in different ways. Ideally, there should be a management layer between the providers and consumers to ensure complete flexibility for implementation protocols.

2.4.1 Transport platforms

When systems are distributed across a network, the single subsystems must communicate and interchange data with each other. In the broadcast environment one may use many different facilities for providing data transfers between systems' components. These facilities can be combined to form a single transport platform (Figure 2.5).

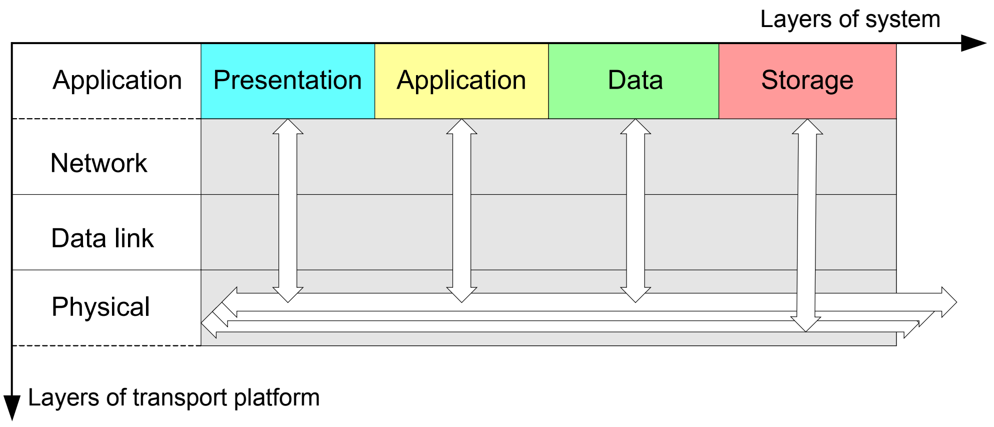


Figure 2.5 Transport platform of a system

The transport platform can be seen as a special component of an integrated system, therefore it may have its own layers (the vertical column on the left), which represent different types of components. The components of the overall system (the right-hand four in the first row) belong to the “Application” layer of the transport platform and make use of its services to interact with each other.

A transport platform provides data transfer between components, which may belong to the same layer or to different layers. At a lower level, a transport platform provides physical connections with other systems.

The current broadcast environment is characterised by a multiplicity of components at the physical layer, e.g. those used for hard real-time, soft real-time and non real-time transfer of media data, Metadata, control signals, timecode etc. Broadband signal inputs and outputs at the presentation layer use the SDI/AES protocol stack supported by coaxial cables. In automated play-out systems, VTRs are controlled via RS422/RS232 protocol stacks, etc.

2.5 Business models for integration

Before specifying any technical details of a system integration architecture, we must clarify the business purpose of integration.

Integrated systems provide a new workflow, which consists of part of the workflows of each (existing) system being integrated. Additionally, new functionality can be provided, which is available only to the final super-system and not to any single system.

Three types of integration are common:

1. corporation
2. federation
3. confederation

The features of each type of integration depend on some criteria: stability of membership, quantity of shared goals and subordination in association.

In *corporation* all systems involved have a single goal, hard defined membership, and are subordinated to one system. Consider automatic play-out as an example: there is a single goal (multi-channel play-out), a hard membership (a set of VTRs and video servers) and one superior system (the device server).

In *federation* each independent system acts with its private goals, but voluntarily gives part of its functionality to a superior system. Membership of the integrated system can be changed without loss of federation functionality. For example: a broadcast network consisting of regional stations, joined with a central station.

In *confederation* each system is still fully independent, with its private goals, but benefits from the integration by exchanging information to improve its business position. However, a system is never subordinated to another member. Membership of a confederation can change very easily. Example: a collection of archives.

2.5.1 Vertical and horizontal integration

The classification of integration types into vertical/horizontal is quite common and useful for describing some typical characteristics of the integration process.

Vertical integration means that integration is possible only via the Presentation layers of the systems. Figure 2.6 shows the vertical integration principle. The association of vertically integrated systems forms a new system, a “vertical system”.

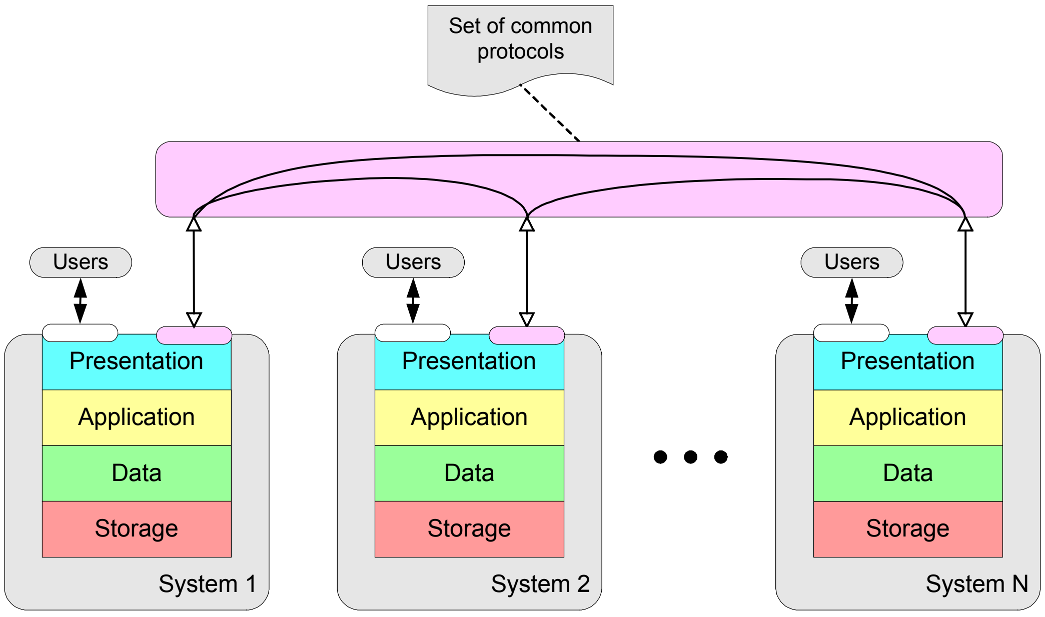


Figure 2.6 Vertical integration

Horizontal integration means that integration is possible not only via the Presentation layer of each system, but via other layers as well. This is shown in Figure 2.7.

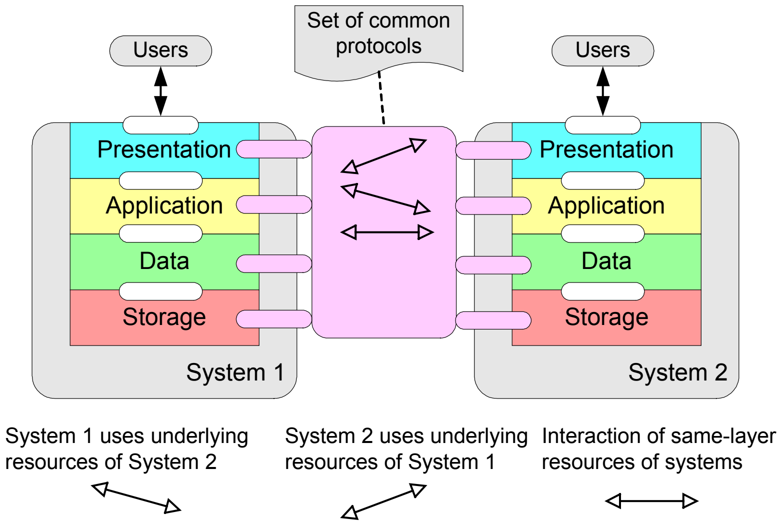

Users
Set of common protocols
Users
Presentation
Application
Data
Storage
System 1
Presentation
Application
Data
Storage
System 2
System 1 uses underlying resources of System 2
System 2 uses underlying resources of System 1
Interaction of same-layer resources of systems

Figure 2.7 Horizontal integration

Horizontal integration can involve only systems that are open also at their intermediate layers. The association of horizontally integrated systems forms a new system, a “horizontal system”.

Figure 2.6 and Figure 2.7 are focused on the logical aspects of interactions, while the mutual understanding at a physical level is provided by a system's transport platform. (The layers thus don't talk directly to each other, but always via the transportation level.)

2.5.2 Where does Middleware come up?

There probably is no middleware definition agreed by the whole IT community. The term 'middleware' is used extensively in technical discussions, but often lacks a clear definition.

For example: "Middleware is software that connects two or more otherwise separate applications across the Internet or local area networks.", "Middleware is an evolving layer of services that resides between the network and applications", etc.

Application users and programmers see everything below the API as middleware. Networking gurus see anything above IP as middleware. Those working on applications, tools, and mechanisms between these two extremes see it as 'somewhere between TCP and the API', with some even further classifying middleware into application-specific upper middleware, generic middle middleware, and resource-specific lower middleware.

In summary, there are many views of what middleware is. Given such a dynamic morphing nature of middleware, a consensus from the broadcast community is required to identify some core middleware services.

In general a middleware component can be a piece of software (e.g. a code library, a distributed service) useful to solve a particular integration problem. The mission of a middleware component is to adapt one or more of the features of two interacting components in an integrated system, namely:

- semantics
- I/O operations and parameters
- behaviour

To identify and specify the boundary of any middleware component, it is a useful strategy to concentrate on the domain in which integration of systems takes place. So, in the context of our broadcast environment we have to concentrate only on broadcast specific middleware.

Layers and components
It is interesting to investigate the role of middleware components in relation to the layers. As shown in Figure 2.8, middleware is 'the part in between', allowing the layers, modules or applications to be collected in different presentations or views, or allowing the use of the same modules of logic, or sharing of the same data across systems. The necessary communication flows through the middleware components' interfaces.

It is important to notice that, given a certain layered view, valid for a certain class of users, only a subset of all middleware components is relevant to these users. E.g. when working with a broadcast-specific payload of message exchange, only the components concerned with the specific form of these data are relevant. This notion is particularly useful when designing an integration project.

Middleware components facilitate information exchange between in-system objects and unify access from out-of-system objects.

For example, an application layer component accesses a middleware component when we try to extend an application for personalization (customisation). Personalization logic usually involves three different aspects: the rules (behaviour), the user profile, and the application of those rules (access operations) to particular business objects (semantics).

To accomplish this, in the integration phase we recognise the need for a technique to intercept messages that are sent to an application object, so that execution of the methods related to those messages can be altered in a way that does not impose changes in the application domain of the system.

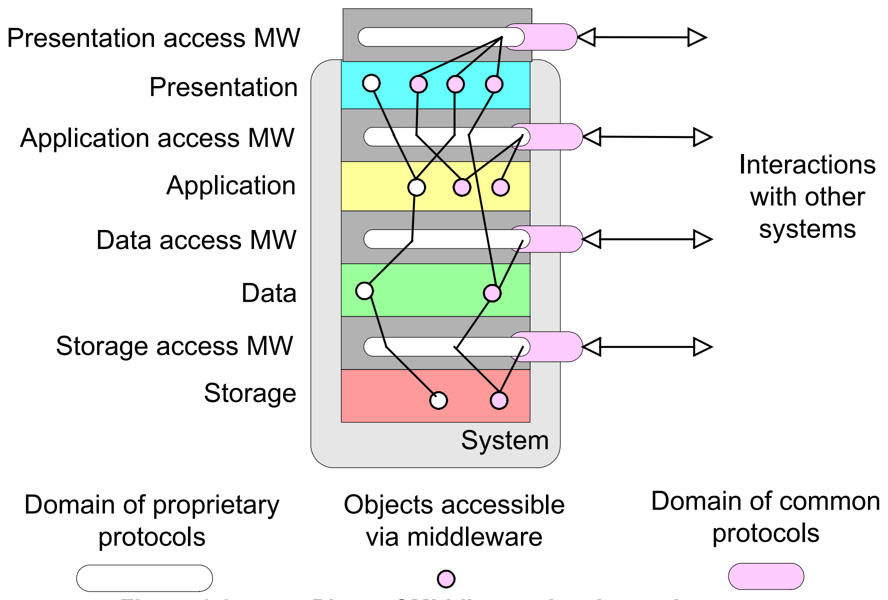

Presentation access MW
Presentation
Application access MW
Application
Data access MW
Data
Storage access MW
Storage
System
Interactions with other systems
Domain of proprietary protocols
Objects accessible via middleware
Domain of common protocols

Figure 2.8 **Place of Middleware in a layered system**

Depending on the complexity of the systems involved, middleware components may be seen as a complex set of executable modules, data tables and other facilities with their own architecture and infrastructure. Thus, middleware components can be represented as a separate system (Figure 2.9).

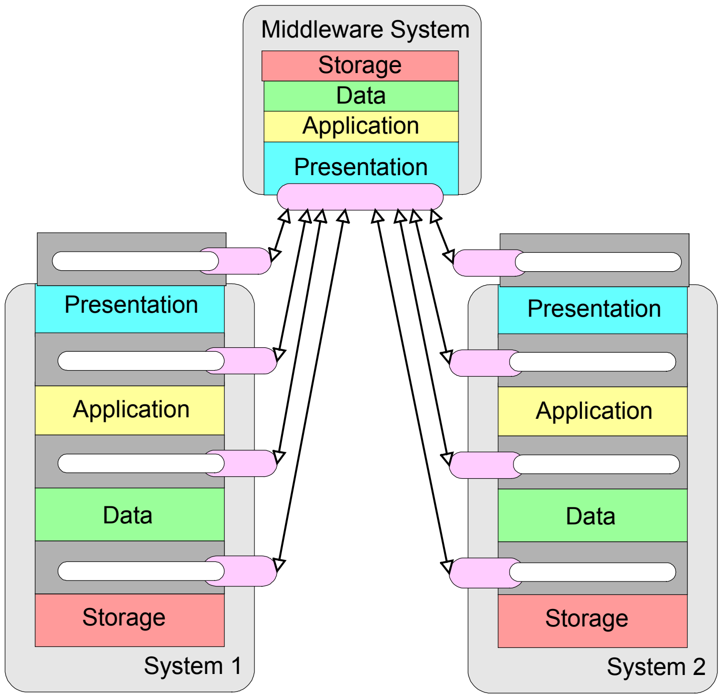


Figure 2.9 Middleware as separate system

The middleware system works as a glue between the layers or across systems. Some parts of the middleware system can run in a separate process space implemented in a separate extra system, while some other parts can run in those systems involved in the integration. The relation between the volume of centralised and distributed functionality may be taken as a measure for characterising the degree of middleware system distribution. There are two marginal cases:

1. Fully distributed middleware exists in associated systems without any centralised facilities. The number of connections equals **N x (N-1)/2,** which is the same as for the vertical integration approach discussed earlier.

2. Fully centralised middleware exists when it is realised strictly in a separate new system. This introduces the 'hub and spoke' paradigm, with the central component facilitating all interactions between the subsystems. The number of connections then equals **N.**

2.5.3 Communication middleware

The IT definition of middleware can now be interpreted coherently. As Figure 2.10 shows, communication middleware provides the adapter components which allow for interconnection and communication between the other components.

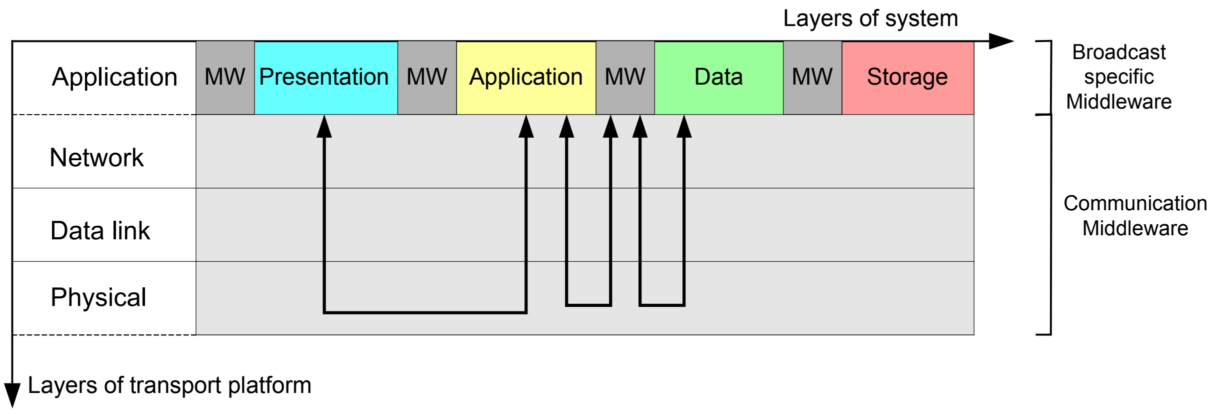


Figure 2.10 Middleware domains

In the example in figure 2.10, objects from Presentation and Application layers interact with each other directly, but another object from this layer can interact with Data objects only via a middleware component. The middleware thus effectively 'translates' functionality and data between the objects with different interfaces.

2.5.4 Middleware shifts System Integration paradigm

The middleware concept introduces a new type of system integration – integration based on services.

As further developed in later sections, services are components that are transversally useful in more than one integration problem. They have an interface that is often useful in common integration tasks. An example is an "authentication service", that is, a component providing authentication and access information to a set of resources based on a specified login.

Extending the concept to a wider application makes it possible is to have a whole service-orientated way of achieving integration: a Service Oriented Architecture (SOA).

A SOA defines how two computing entities interact in such a way as to enable one entity to perform a unit of work on behalf of another entity. The unit of work is referred to as a service call, and the service interactions are often defined using a description language. Each interaction is self-contained and loosely coupled with others, so that each interaction is independent of any other interaction.

Another typical aspect of such an architecture is the protocol independence. The protocol independence of SOA means that different consumers can use services by communicating with the service in different ways. Ideally, there should be a management layer between the providers and consumers to ensure complete flexibility regarding implementation protocols.

This scenario offers a new integration paradigm, in which each system performs a part of the general task of integration, and a system “glue” concentrated in a separate component provides the "brain” and “nerves" ensuring the overall system integrity.

2.5.5 'Bread & butter' services

Each company has to define, describe, and maintain a catalogue of the super-services in use, e.g. for web-services this would be a WSDL repository. The 'Bread & butter' services represent the broadcast- specific functionality to be provided by any integration architecture. In the next main section are discussed arguments and proposals for agreeing on common definitions among broadcasters with regards to a broadcast-specific service architecture.

2.5.6 Pervasive Services

Pervasive services are those services not limited to use in the broadcast domain. They perform actions which could be of interest to any IT system. Therefore we do not have to create them ourselves but only to select those technologies which are applicable to the broadcast industry. A basic classification follows:

<u>Message Transport Services</u>

Message delivery is an issue for any information system. IT-based TV production systems are basically information systems. There may be some broadcast-specific requirements for the message transport. For this reason it would be beneficial to define QoS requirements depending on what a message is used for.

<u>Time Reference Services</u>

Each traditional broadcast studio has a reference timecode. This is what this service is used for. It provides a synchronised time reference for the whole system.

<u>Transaction Security Services</u>

A middleware-based system has to provide a mechanism to undo transactions. Even if those transactions have an impact on different distributed objects all over the system.

<u>Allocation Mechanism Services</u>

Some resources are unique in an-IT based TV production system; others are limited in use by concurrent users. This has to be managed.

<u>Authentication Mechanism Services</u>

There needs to exist one instance within an IT system that has knowledge of the resources and their permissions. An identification and validation mechanism for this is required.

2.5.7 A technology tip: web-services

Any task of integration is concerned with the orchestration of a group of specialised services in support of a single business process. From a technical perspective, Web-services are a standardised way of integrating Web-based applications using open standards, including XML, SOAP, WSDL and the UDDI specification.

Simply put:

- XML structures the message
- SOAP transfers the message
- WSDL describes the available services
- UDDI lists the available services

XML describes both the nature of each self-contained function and the data that flows to and from it.

Web-services must conform to a particular specification format to ensure that each service function will easily integrate with others to create a complete business process. This interoperability allows businesses to dynamically publish, discover, and aggregate a range of Web-services through the Internet to more easily create products, business processes, and value chains. Typical application areas are B2B integration and content management.

While the development and deployment of Web-services does not require an underlying technology platform, two different Web-services architectures - Microsoft .NET, and Sun Microsystems J2EE - are competing head-to-head to become the preferred platform for Web-services developers.

XML schemas will be central to how organisations communicate and the unique characteristics of Web-services suggest new design patterns. These patterns include messages with a high degree of granularity, asynchronous messaging, bi-directional services, endpoint discovery, and protection against the mishandling of messages received more than once. Web services standards permit application-to-application interoperability, but the co-ordination of a set of Web-services working toward a common end is an open issue.

3. Middleware services for broadcasters

As discussed in section 2.5.4 a Service Oriented Architecture provides a good basis for a system architecture. As services and their specifications are at the core of a Service Oriented Architecture we will start with a definition of the term *service* itself.

3.1 Definition of a service

A service is a software agent that performs some well-defined operation (i.e., “provides a service”) and which can be invoked outside the context of a larger application. While a service might be implemented by exposing a feature of a larger application, the users of the service need only be concerned with the interface definition of that service.

The question “What does a software agent (service) do?” occurs regularly when working on service specifications. To help answer this question we decided to describe how one might develop a Service Oriented Architecture in order to provide a basis for system specification. The service architecture is specified by the means of a model. This gives vendors the opportunity to specify how their products can perform, i.e. what services they expose.

3.2 Business Objects

Usually services process data: think for example of a transcoding service that operates on a data stream. This data reflects objects in the real world and which are related to the *business* for which the service has been specified. For this reason data *objects* specified in this sense are referred to as *business objects*.

Business Objects don't have to be data, they may also be tools such as transcoders, player applications, etc., or logical objects such as an audio track, a video track, the configuration settings of a device, etc.

3.3 Types of services

As we discussed in the Introduction (see section 1.8) there are three types of services:

- Super Services, services supporting business processes
- Broadcast specific services
- Pervasive services, services with general applicability

It is useful to define boundaries in order to specify which aspects are covered by an architectural model. For this reason we use the Model Driven Architecture as a declarative concept (see chapter 4.3). Since MDA is an architectural approach this would lead to a service definition that can be used as a Domain Model according to MDA. We call it the Broadcast Domain Model.

So which middleware functionality is relevant for broadcast systems? To identify these, we worked out a methodology. It is based on the usual modus operandi in business analysis. The essential aspect of a service is the functionality it provides, this can be found by answering “What to do?” questions. The characteristic feature of this methodology is the consecutive application of the “What to do?” and the “How to do?” questions.

Thus, the first step in the design of service-oriented architecture is to identify what functionality is required. A detailed description of an analysis using this methodology may be found in annex D.

3.3.1 Super Services

In order to support complex business processes you can employ collections of services by the means of orchestration. Since business models and business processes among different broadcasters may vary, super services are not necessarily transferable.

3.3.2 Broadcast specific services

A broadcast domain model should include all service definitions specific to the broadcast domain. This is where the EBU and its members have some core competence. This gives the possibility to assign responsibilities very precisely.

The responsibility for the definition of the services is with the standardisation bodies. The responsibility for the compliance of the products implementing certain interfaces is with the vendors and is verifiable.

Examples of services specific to the broadcast domain may be:

- Stream Control
- Transfer Control
- Transcription
- Face Recognition
- Shot Detection
- Transcoding
- Metadata Operations
- Playout Scheduling

Other Domain Models

In other domains, models have been developed as well, just as it appears to be necessary in the broadcast domain. This can help re-use solutions in the broadcast domain for problems which are not specific to the broadcast domain, e.g. cost-centre management, billing, load balancing, mirroring etc.

It may be that descriptions of the same concepts exist in different domains. This could lead to ambiguous definitions. In order to prevent this, UML defines the concept of *packages*. It enables the architect to unambiguously define which representation of the concept he/she decided to use.

3.3.3 Pervasive Services

Pervasive services provide functionality that is not specific to the broadcasting needs. The use of pervasive services gives the opportunity to benefit from solutions worked out in other industries. It may be necessary to investigate the appropriateness of a certain technology and related pervasive services in order to support broadcast specific services dependent on these. We don't think broadcast bodies, such as the EBU, should specify any pervasive services.

Pervasive services are reused by other services and applications. This means the responsibility for the functionality of pervasive services is with the broadcaster and with the vendors of the pervasive services, depending on the configuration, application or implementation issue that is being addressed.

Examples of services not specific to the broadcast domain are:

- Message dispatching
- Services registry
- Resource management

- Workflow management

3.4 How to specify a service

The only criteria for us when specifying functionality are the semantic relationships. The final assignments may differ, depending on the objective of the architect.

3.4.1 General description

The general description of a service should provide a detailed description of what the service is intended to be used for, e.g. what it is for, why it is needed.

3.4.2 Purpose

The purpose of a service should be a short description of which functionality the service exposes. This information should be a more abstract description of the interfaces. It may be used by system integrators who do not implement the service but select the appropriate service in order to solve a problem. The purpose of a Metadata operations service could be simply “Metadata retrieval and Metadata manipulation” while the “interface” description and the “general description” of the service need to be much richer.

3.4.3 Interfaces

Interfaces are variously called functions, methods, actions or operations in some implementation contexts. They provide access to the functionality of a service. Interfaces have a static definition; this is often referred to as their signature. The signature defines the parameters passed to, and received from, the services by use of the interfaces.

They also have behaviour, which is the dynamic characterisation of the interface. If a service is used (by the invocation of an interface) the state of the service may change. For example, a resource is allocated and the service that manages the resource may lock the resource and block subsequent requests for that resource.

3.4.4 Inclusions

Since application of the Service Oriented Architecture for broadcast leads to improved interoperability, the components representing the Broadcast Domain Services must share common concepts, such as common data types.

Definitions of these concepts are required and have to be included in the implementation of the services, for example in a library defining Business object definitions for broadcast. The inclusions' relation to other specifications is of interest for developers. It is information that is required in the design and implementation phase of a service's development.

Business object example

Modern play and record devices for real time essence streams, usually work internally with a frame count on a certain frame rate. However this might be very difficult and error-prone for human beings. For this reason GUI and dedicated control panels usually use Timecode values for specifying in- and out-points, duration, offsets, etc.

A recurrent problem in information exchange among broadcast systems is the description and calculation of time values. The purpose of the TimeValue business object is to ensure accurate calculation and system behaviour with regard to time-related operations.

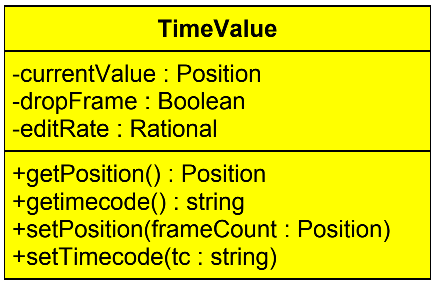


Figure 3.1 Part of the specification of the TimeValue business object

Structure

The static structure of the Framecount business object consists of three values. These three values specify a Framecount and the calculation of a position.

dropFrame

Specifies whether frames have to be dropped for calculation of the Timecode format, or not. True indicates frames have to be dropped while False indicates that frames do not have to be dropped.

currentValue

Indicates the current frame count of a TimeValue instance. This may indicate duration, an offset, or any time value for a real-time essence-stream related operation.

editRate

Specifies the base for the calculation of time values.

Behaviour

The TimeValue business-object possesses different “setter” and “getter” methods in order to adjust and retrieve the internal frame count state of a TimeValue instance. Since timecodeBase and dropFrame may not change during the lifetime of a TimeValue instance, these values shall only be defined in the constructor method of the TimeValue business object.

setTimeCode(string tc)

Sets the internal frame count of the timeValue according to the passed tc string value using format “hh:mm:ss:ff”.

string getTimeCode()

Gets the current frame count as a tc value in format “hh:mm:ss:ff”.

setPosition(Position frameCount)

Sets the internal frame count of the timeValue according to the passed parameter.

Position getPosition()

Gets the current frame count of the TimeValue instance.

3.4.5 Dependencies

A service may use other services. It may not be able to fulfil its purpose if they are not available. The dependencies' relation to other services is of interest for developers and integrators. It is information which is required in the design, implementation and integration phase of a service development.

3.4.6 Categorisation

A service may be categorised as being of a certain type. For example: event-based, control-based, monitoring-based, data transfer-based. Also qualifiers, like async/sync, one-to-many/multicast, unidirectional/bi-directional and non real-time/soft real-time/hard real-time may be used. It needs to be investigated what categorisations already exist.

3.5 Different views

There are different types of actors involved in the development of an IT-based TV programme production system. Three different types have been identified in the study so far:

- Planning engineers
- System integrators
- System developers

Each of them represents a different role. The different tasks of these roles require different views on the modelled system. For this reason we propose to include different views in the specification of the service architecture. Of course the different sections of a service architecture specification may be of interest for anyone involved role in system design, integration and development, but the different views should reflect the most useful abstraction level for different roles in order to perform their task.

3.5.1 Planning engineer view

The view on the systems required for planning engineers is a UML Package Diagram. The UML Package Diagram consists of different packages that contain collections of service definitions. These collections represent the choreography of services that is recurrent in a specific context, e.g. Ingest.

The idea of putting services together in a package in order to give some guidance in solving recurrent problems is very similar to the pattern approach that is already applied in other industries.

Since the Package Diagram provides some information about the use of the services in a broader context, the description of a package should be a specific document.

3.5.2 System integrator view

The view on the systems required for system integrators is a UML Component Diagram. It provides a description of how to use a service. It has to represent the interfaces of the service from a high level view and the dependencies. The deployment diagram should be part of the service description.

3.5.3 System developer view

A system developer needs very detailed information about the components to be implemented. For this reason the service specification of a service needs to contain descriptions of the dynamic behaviour of the service as well as of the static design.

In order to describe the static design, a service UML Class Diagram shall be used. It contains a description of the interfaces of the services as methods of the classes. The description of these methods shall be the complete signature of the method including input and output parameters.

The description of the interface shall include, depending on the characteristics of the interfaces, UML Sequence Diagrams and/or Activity Diagrams. At least one diagram for each interface shall be provided.

3.6 Summary

In order to make use of the middleware concept, the definition of a domain model appears to be useful and necessary. This kind of work has already been undertaken within other industries, such as health care, the automotive industry, etc.

A domain model has to describe functional elements used in order to solve problems within a business domain, in our case the broadcast domain. It should not describe the internals of functional elements nor each detail of those, but does have to enclose the mission critical aspects. The functional description should be limited to the description of the interfaces and should include the static definition as well as the (dynamic) behaviour.

This may be achieved by application of the concept of the Service Oriented Architecture. The availability of a service description limits the risks for the vendors as well as for the users. The complete description of a service-oriented architecture that represents the Domain Model for broadcast consists of two different types of specifications:

- The Package Diagrams provide a description of standard solutions for recurrent problems in the broadcast domain. They do not provide a detailed description of the services used.

The Service Description provides a detailed description of a single service. It also provides a detailed description of the service's interfaces. It does not describe the internal implementation of the service.

Figure 3.2 shows a possible architecture for broadcast systems based on SOA. The SOA itself may be derived from concrete business processes that have to be refined to a higher level of abstraction in order to get transferable service definitions. This process is described in annex D in detail.

The resulting service definition may be used in compositions that will be defined by applications. Building these applications supports Business Processes in a very efficient way by the reuse of existing technology for recurrent problems.

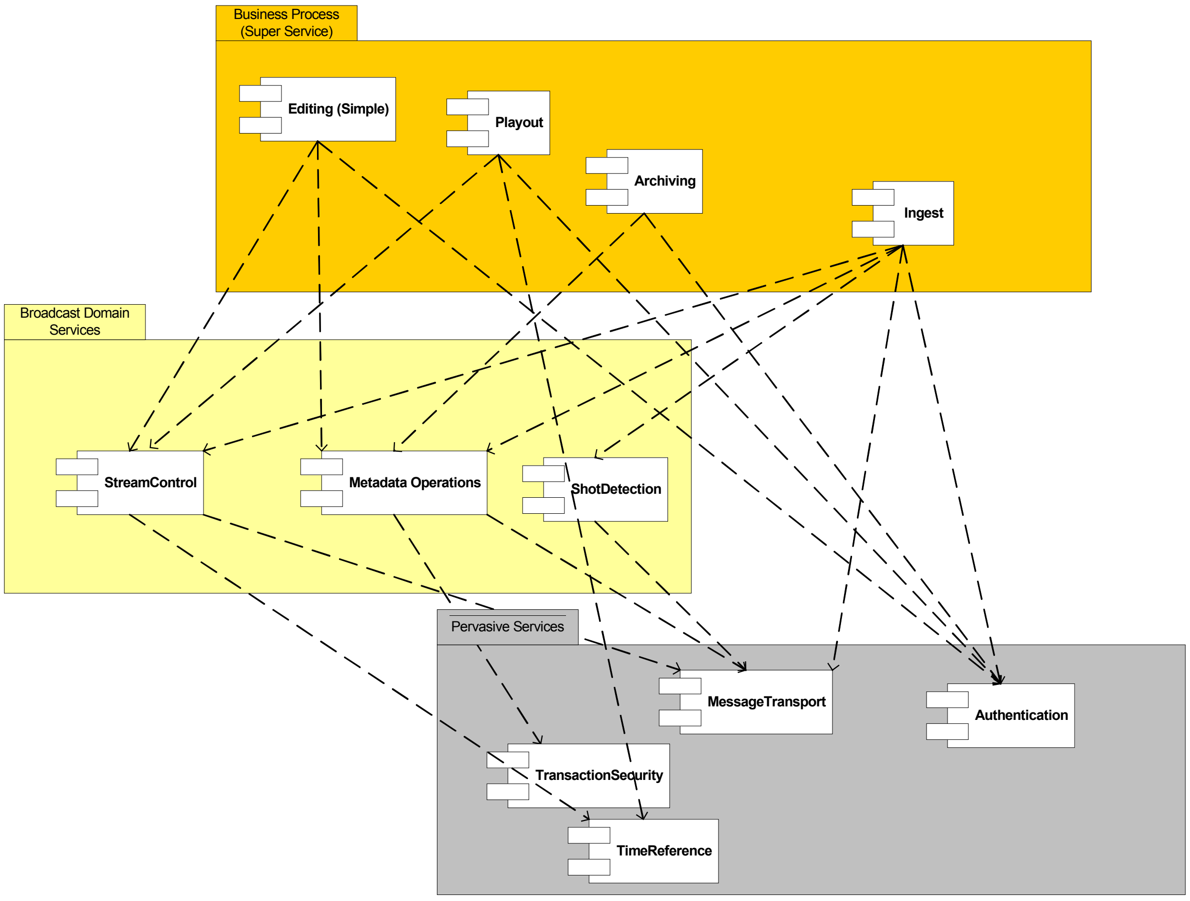


Figure 3.2 Overview showing how the common Middleware services relate to each other

4. Modelling in the broadcast environment

Within the broadcasting industry there is a tremendous change happening, from dedicated broadcast hardware to IT based technology. For this reason the impact of software technology on the broadcast industry is increasing.

The use of models in order to describe technical systems is not a new thing. As well as UML diagrams, block diagrams and connection schemes are models. The difference between UML diagrams representing software systems is that there is not necessarily a physical object represented by a specific symbol in a diagram.

UML has been developed and specified by the Object Management Group (OMG), which is a non-profit organisation of more than 1200 companies and individuals. The members of OMG are active in many different industries: some of them in the broadcast domain.

4.1 Unified Modelling Language

Modelling is a widely-used method in IT system design. Arguably, the best known modelling methodology is called UML (Unified Modelling Language). In the last 10 years it has established itself as the predominant formal notation for the description of IT projects.

UML provides different types of diagram for the static design and the dynamic behaviour of IT systems.

- Use Case Diagram
- Class Diagram
- Activity Diagram
- Package Diagram
- Sequence Diagram
- Object Diagram
- State Chart Diagram
- Collaboration Diagram

In the following we will give a short introduction on which diagram to used in which case. Unfortunately, an in-depth description would exceed the objective of this document.

Our example for the description of the different UML diagrams is the Record process of a VTR. Our task is to describe different aspects of this system using UML.

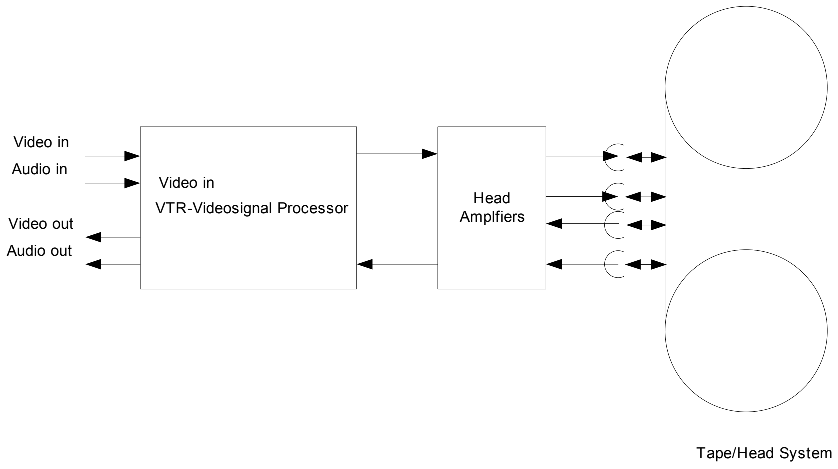


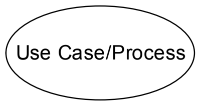
Figure 4.1 **Example video system to be modelled**

4.1.1 Use Case Diagram

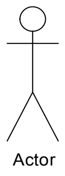
It takes some knowledge of software development to understand all types of UML diagrams. However, if you are not a software developer you will not need to understand all of them. The most important diagram type for people who are not software developers is the Use Case Diagram.

A Use Case Diagram gives an overview of what happens in a business process. It does not explain how the result is achieved, but only what happens. Use Case diagrams are used to describe the collaboration of users and systems in an IT-based environment. They provide an overview of the required processes and interfaces but not a detailed description.

There are two types of symbols used in Use Case diagrams:



This may be a process or a Use Case diagram itself. A Use Case consists of one or more processes but may also be a collection of Use Cases. UML does not distinguish between 'sub Use Cases' and processes. For this reason, nor will we. We refer to this symbol in the text as a process but we call the diagram a Use Case Model.



An actor may be a person or a system. An actor is a role of anything which is acting in a Use Case and manipulating the system from the outside.

In a Use Case Diagram you will find information about the actors involved (actors are not necessarily human beings, but also may be systems, tools, software applications, etc.). Usually you will also find the processes and the associations between them.

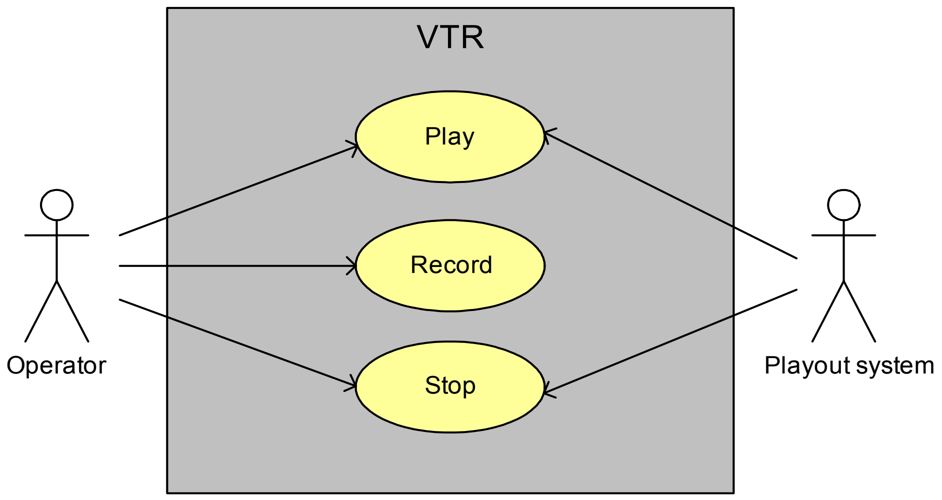


Figure 4.2 Use case diagram for use of the video system.

A detailed example Use Case model for TV Programme Production can be found in annex D.

4.1.2 Class diagram

Modern software is usually developed following the Object Orientation paradigm. The Class Diagram gives an overview of the class structure of a piece of software. Classes are the declarations and implementations of real world objects in a programming language (e.g. C++, Java, etc.). Usually the Class Diagram will show the classes and the associations between the classes. You will not necessarily find all associations of a class in one Class Diagram. It depends on the objective of the Diagram as to which associations are shown and which are not.

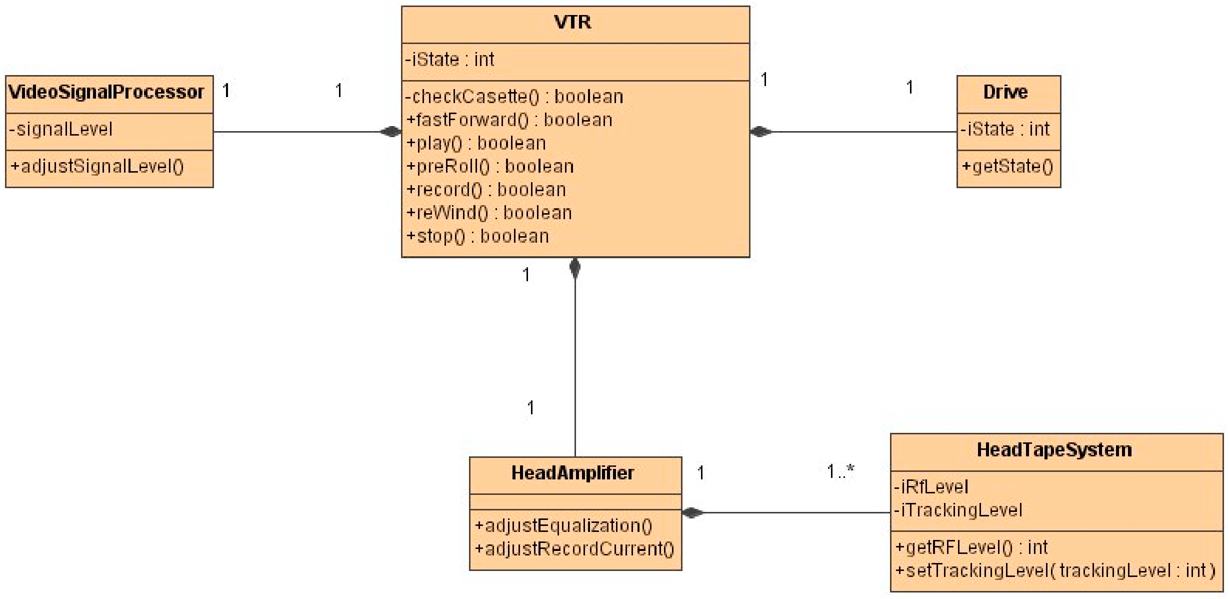


Figure 4.3 Class diagram

4.1.3 Activity diagram

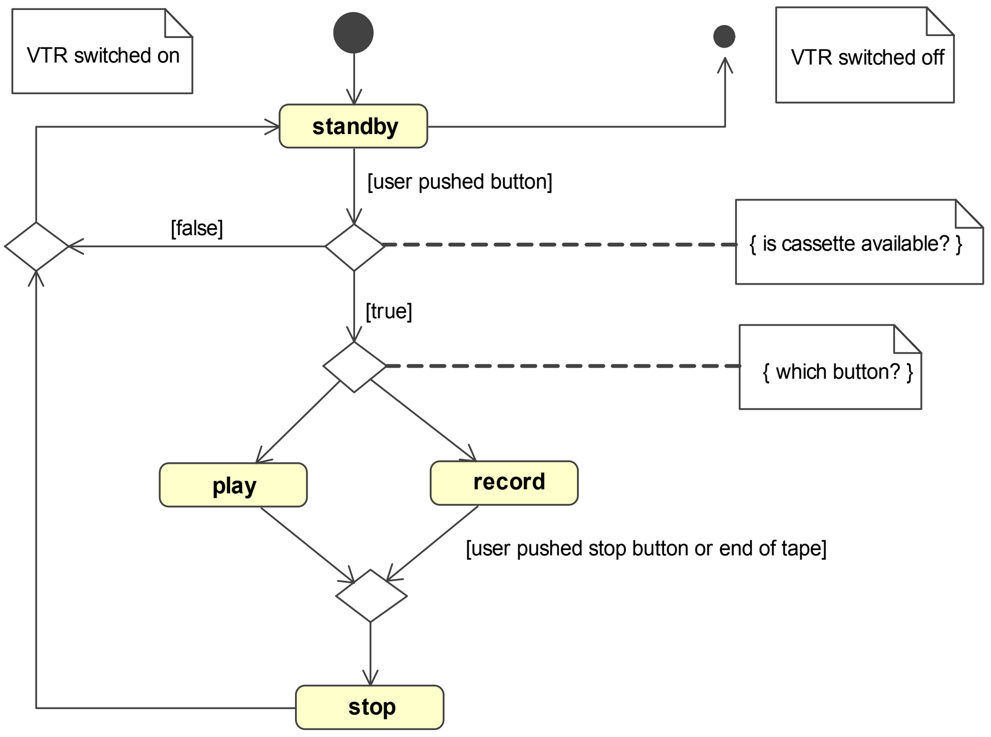


Figure 4.4 Activity diagram defining behavioural aspects

If the concept of a state machine is used in a project, the behaviour of the state machine could be described using the Activity Diagram. In the Activity Diagram you can see the different states of the state machine. You may also see which state follows which and you can see what the conditions are for a state change.

4.1.4 Package diagram

Packages encapsulate model elements that belong together for some reason. The reason may be technical or semantic, depending on the intention of the designer. A model element (e.g. class, interface etc.) belongs to one package at most. The idea behind the packages is to ease the overview of the model, or of parts of the model, and to provide a concept for namespaces. A package can include just a few classes, a complete subsystem or a model of an entire application domain. Between packages, different types of relationships may exist.

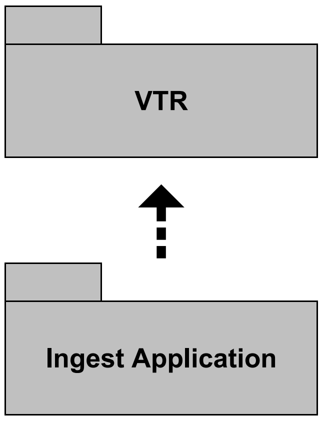


Figure 4.5 Package diagram defining the organisation of the model

Figure 4.5 shows a package diagram that defines a dependency relationship between two packages. Changes in the package 'VTR' also require changes in the package 'Ingest Application'. Further note that the Ingest Application makes use of parts of the VTR package or even of the entire VTR package.

4.1.5 Sequence Diagram

A sequence diagram describes one process. It is very similar to a flow-chart and describes how the process proceeds. In the Sequence diagram you will find the different objects that are created for the different classes. You may also see what the lifetime of an object is, this being how long the object exists in the context of an application. Additionally, you can see which object has control of the process at any given time.

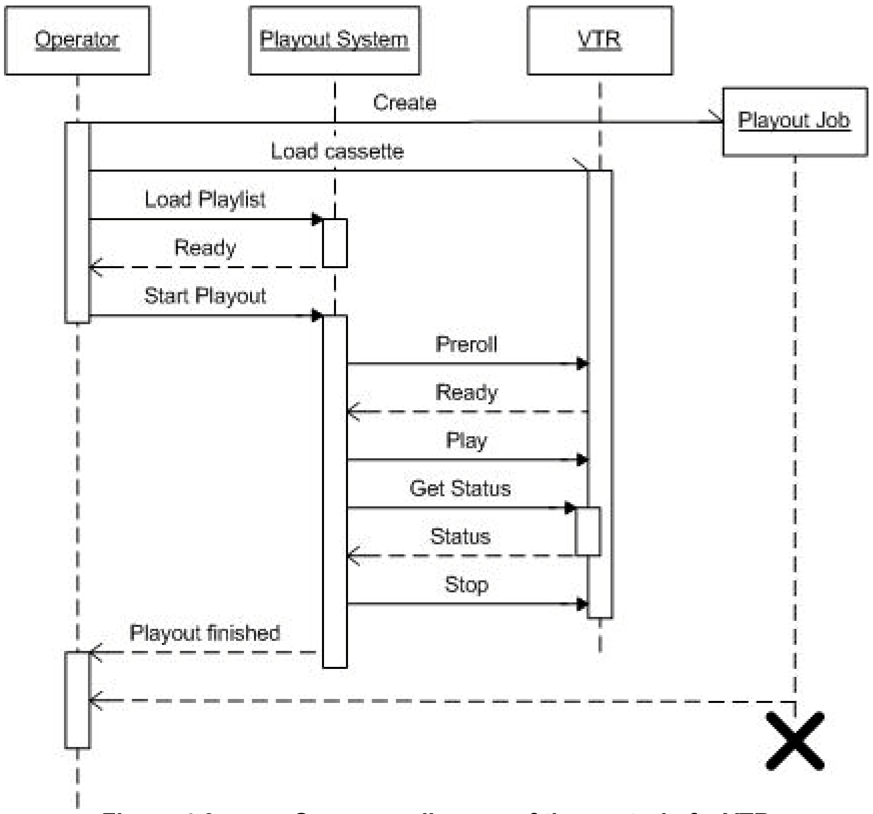


Figure 4.6 Sequence diagram of the control of a VTR

4.2 More UML diagrams

4.2.1 Object Diagram

If Object Oriented software is executed, the Class Model 'comes alive'. In the memory of the computer, objects are created. (In terms of software development they are said to be 'instantiated'). The associations of those objects are represented in an Object Diagram. The Object Diagram specifies a certain operational condition at a certain time.

4.2.2 Collaboration Diagram

Collaboration Diagrams are an alternative to Sequence Diagrams. They contain objects and the associations between them. The order of execution of the operations is visualised by the numbers of the operations.

4.2.3 State Chart Diagram

A State Chart Diagram is a simpler description of a state machine than an Activity Diagram. The limitations compared with the Activity Diagram are that a State Chart Diagram cannot provide information about the coupling of the state and a process.

4.2.4 Summary

UML provides al lot of different diagram types in order to describe the architecture as well as the behaviour of a system. Unfortunately, it does not give much guidance on how to use the diagrams or on the minimum level of detail that has to be modelled. There are no mandatory elements within a UML diagram. This means that the same system may appear very different if modelled by different people.

Pure UML modelling is generally not sufficient for the documentation or specification of an IT system. Some textual description is still necessary. Additional to the textual description and diagrams a legend should be provided.

4.3 Model Driven Architecture

Model Driven Architecture is an architectural approach that splits off application domain and implementation platform issues.

Therefore MDA uses the concept of Platform Independent Model (PIM) and Platform Specific Model (PSM). The PIM as well as the PSM may be specified by the use of UML. MDA has been developed by OMG as well as UML.

4.3.1 Platform Independent Model

A Platform Independent Model consists of a Domain Model and the Pervasive Services. The use of Pervasive Services is not limited to any application domain; it may be that of broadcasting but equally health care, automotive, etc.

The Domain Model is used to describe the domain specific issues of a system. To get a structured model of an analysed subject it may be useful to mix the approach with the concept of Service Oriented Architecture. This means that the Domain Model consists of a number of different services and their interface description, which will be used to support the services required by the business processes in the investigated application domain.

4.3.2 Platform Specific Model

The Platform Specific Model is created by refinement of the Platform Independent Model. The process for the refinement is defined by the use of UML profiles. The concept of UML profiles is an extension of UML. Modern software design tools perform this refinement automatically.

4.4 Framework for modelling and information, where to host it

From setting the early requirements through the development process and on to implementation, deployment, and daily operation and change management, these models contain the information about many aspects in our systems. From high-level business value-chains and process modelling to detailed descriptions of entities and relations as well as the actual deployment in hardware and infrastructure.

In a complex system environment with a high level of integration, more and more based on separate modules of functionality shared across applications, it is vital to have relevant and up-to-date information about the systems and their relationships. Not least because of the speed of change and development of our systems and their functionality. This stresses the need for hosting and updating information in a structured way that makes it easily accessible for different functions and people in the organisation. Instead of separated documents located in many places, this information can be placed in a common database, often referred to as a *repository*. This can host information, as well as rules and functionality, for change management and quality assurance in the developing process and in daily operation.

As many people are using the same pool of information, several structured ways to address the information and to filter the appropriate set for a given task have been developed. One of the more well known ways is the Zachmann framework, which combines different organisational levels and their tasks with the classical questions about what, how, where, who etc. Some repository systems are even using this as a user-interface for working with the information contained.

This is further described in Annex F. The relationship between Zachmann framework and MDA is specified in [2].

4.5 Summary

The use of modelling in the broadcast domain is not a new thing. However, due to the increasing impact of IT, and especially software technology, new types of models are being introduced. Since the share of software development in the design and integration of IT-based TV productions systems is growing, there is an urgent need to be able to apply the same concepts and tools as in general software development, in the design and specification of the components of IT-based TV production systems.

Another new issue is that there are methodologies based on specific formal description languages, such as UML, which may be applied in the broadcast domain. The intention for the development of those concepts was to cover the complete system development process in order to achieve a more efficient development and more robustness in the resulting system. Nevertheless, even if those concepts are not applied end-to-end, they may be used in order to trace the outline of specification efforts, e.g. standardisation.

The basic use for us of the MDA is to use it as a declarative model; since it is not yet mature it is not useful for us to recommend or declare a process model, so we are using MDA as a declarative model. In chapter 3 you have seen how we propose to apply the concept of MDA in the broadcast domain.

From the use of these descriptions, the need for a systematic approach to store and update the information during the lifecycle of systems or data arises. We would like to end with one final Question (and answer):

4.5.1.1 Do I have to use UML?

Although the use of standardised and formalised techniques is preferred, in our discussions with vendors it became clear this is not an absolute must. The most important aspects to take into account are:

1. Make sure the right people are modelling (the ones who know the work best)
2. Make sure the modelling is clear & unambiguous

So if you'd rather use your own modelling method, there is no reason why it shouldn't work, but if you have the time, try to learn from what is already out there, including UML.

5. Managing System Integration

5.1 Background:

Given the driver that we are forced to work increasingly across systems, departments and locations in an ever-faster changing way, and accepting the basic argument that we must own and understand the definition of our own world (although many parts can be shared with others, hopefully), we have to accept some preconditions:

- Regardless where your organisation is placed on the progression from in-sourcing to out-sourcing of your technology, experience and general recommendation is that it is almost mandatory to keep work on system architecture in-house.
- Successful system architecture and integration is strongly dependant on the ability of the different business areas to define and describe their needs for supporting technologies in a structured and systematic way, which should be largely independent of specific products or solutions from specific manufacturers.
- The strategy should be seen from the corporate business point of view, not just from the engineers' or R&D department's point of view, as these resources and skills do not necessarily exist within your organisation.
- This is similar to Business Process Reengineering - a top-down decision to start working with a system integration architecture as described, but later on, bottom-up work to make it part of everyday operations.
- Dealing with all this increasing complexity and variety is much like working with quality assurance programs: describe what you do and do what you describe.

5.2 The requirements:

To be able to manage system integration architecture to its full extend you will need to deal with:

5.2.1 What and where to implement

- Define business needs for integration (drivers). This could be a catalogue of the most critical workflows that have a lack of integration between the supporting systems. This is true whether the goal is a better support of your workforce and operations or the replacement of manual operations by automation.
- Define and describe the required workflows that are the triggers for system integration in detail, using some structured method (different kinds of process and data-modelling).
- Define the scope and volume of integration in your situation. A certain level is needed to ensure the benefit of implementing a new way of developing and operating your systems, as well as acquiring integration software and hardware.
- Make some strategic decisions about which integration methods and supporting technologies to use in your specific system environment. Or maybe, if strategic changes in this environment are needed to prepare your environment, to be more 'integration-ready' (choice of platforms, standards etc.).
- Make a plan for proof of concept (POC) in your environment, including the first small implementations that could follow it and the subsequent minor real implementations.
- Start with minor integrations in the administrative domain (financial, planning, ERP, etc.) where solutions have been well proven in other areas of business (banking, industrial back-office, stock and warehouse management etc.). Then expand to your core business: broadcasting and programme production, as these areas are still newcomers to this field.

5.2.2 How to manage

- Realise that, for most of us, this is very much an organisational change management programme and less a routine technology purchase project. So set up and prepare your method for dealing with changes between people and in your management structures.
    - This includes the need for different skills in your workforce around the development and operation of your systems.
    - As systems are used increasingly across different departments and locations new management roles are needed to make cross-functional decisions: System Owners.
    - As work processes too are increasingly carried out across departments and different functional areas, more in line with your value-chain than with your organisation, some process-aligned decision mechanisms are needed: Process Owners.

  - As you increasingly define your business objects and Metadata from your requirements instead of struggling to fit into a given model from the manufacturers, you must have skilled people to host and maintain the most critical definitions of Metadata. Even when these are based on common standards, you need to have a dedicated implementation in your organisation: Metadata Manager.
  - If not already in place, you will definitely need system (IT) architects to manage and review both the implementation of integration software and hardware, as well as each project in your organisation dealing with system integration
  - Depending on the size of your internal R&D operation you might need to add to, or expand, your skill base regarding system integration techniques. Or at least ensure that that expertise can be available.
- Ensure the long-term financing of this work. When moving from one-to-one project-oriented system integration to a hub-and-spoke method, it is inevitable that the first projects in the line will bear much of the cost from which the following projects will benefit. Short-term cost/benefit analysis will not work. Aspects of IT-governance and the move from project management to managing programmes of projects are welcome.
- This is very much about keeping an overview in a more and more complex and changing environment. So set up and prepare your methods for hosting and updating information about your environment: Processes, systems, business-objects, Metadata-schemes, data, integrations etc. both in the implementation of your system integration architecture as well as in the ongoing development and daily operations of your portfolio of systems.
- Use your existing change management process (or get one) to deliver both systems and integrations through development into operations and maintenance. The need for an overview is equally valid in the operations domain and you might also need tools for hosting and updating information about the actual deployment and performance of, not only your systems, but also your integrations (brokers, adapters, web-services).

Make explicit, formulated, policies and standards for what you do and what you don't do, both for the methods, technologies, platforms and products you do use and also for those you don't. Not only as an internal instrument, but equally as a tool for communication with vendors and for setting up consistent requirements for your solutions across all of your projects. In this way you help to get the systems ready for integration.

5.3 Setting up an Integration Competency Centre (ICC)

The cost for system integration can differ dramatically depending on how the work is managed. At one end of the spectrum every integration is a stand-alone solution with only one person capable of handling development, maintenance etc. At the other end there is an corporate integration architecture implemented.

To boost economy, quality, capability, etc there is a need for standardisation and streamlining of the integration efforts. To manage and optimise the integration efforts it is important to have a strategic scope that spans more than just the current projects. This governance has two sides - the business decisions side and the operational side, - which carry out the actual development and implementation of the integrations. The latter could be centralised within the company. Gartner has named such a function - Integration Competency Centre (ICC) [3].

5.3.1 Organisation

The size of, and services offered by, the ICC would vary from company to company. A minimal ICC could have the mission to develop supporting guidelines and suggestions. A more extended ICC would govern all integrations, technology, standards, implementations, etc.

If the enterprise has a centralised IT organisation there is an opportunity to establish such an ICC due to the immediate possibility to gain mandate. In this scenario the integration platform will be the enterprise integration backbone where reuse, economy and effectiveness are very important attributes. Depending on the missions of the ICC, the number of people involved can differ significantly from just one person to more than a dozen.

5.3.2 ICC responsibilities

The ICC should be responsible for establishing and maintaining enterprise integration standards, architecture, components, patterns and processes. The most important being:

- documentation standards
- the process of promoting code/components to production
- integration best practices

- the enabling of different techniques for interoperability, e g JMS, MQ, FTP and Web-services
- naming standards
- integration security
- architectural patterns for different integration scenarios
- the Enterprise information model
- reusable integration components. Examples are logging, security, monitoring, system access, data objects (e g XML schema) and protocol components (e.g. for HTTP).

The ICC also acts as an enterprise expert group on integration techniques and tools. To develop and maintain the expertise there is a need to keep up to date with the current and emerging technologies and methods. This includes contact with the supplier of the integration platform, product education, review of new products, reading of white papers, contact with other skilled integrators, reading of standard proposals, etc. With these skills in the ICC there is an opportunity to perform in-house education, act as project member, develop the documentation, etc.

5.3.2.1 Roles in the ICC

Typically the following functions and competencies exist within the ICC. Depending on the size of the ICC, one person could have several roles and vice versa.

- ***ICC manager*** – Management and governance of the ICC.
- ***Project manager integration*** – Project leader according to ICC instructions.
- ***Information modeller*** – With the skills to create and maintain information models.
- ***Integration product expert*** – Knowledge on the integration products used for integration.
- ***Architect*** – Responsible for developing and maintain the integration architecture.
- ***Hardware/OS expert*** – Knowledge about the platforms that is used for the integration products.
- ***Developer*** – Development of specific integrations and supportable components (e.g. adapters).
- ***System administrator*** – Operation, configuration and other management of the running integration solutions.
- ***Tester and QA responsible*** – Responsibility for QA and testing.

It is important to involve the right people in your ICC and make them a team. Try to find people who know about the business and applications already and who have good contacts with the staff in the project teams [4].

5.3.3 Responsibilities outside the ICC

It is of vital importance to ensure the solutions are verified and accepted by the business units and users who have to live with this highly integrated environment: they should feel better and more flexibly supported.

Nevertheless, they will have to understand and accept the trade-offs between limitations due to rational behaviour and standardisation on one hand, and the appropriate level of flexibility and speed on the other.

Regardless of whether the systems and their integrations are supplied by internal or external suppliers, there has to be a balance in the management between the internal customer and the supplier or vendor. This balance could be illustrated as follows:

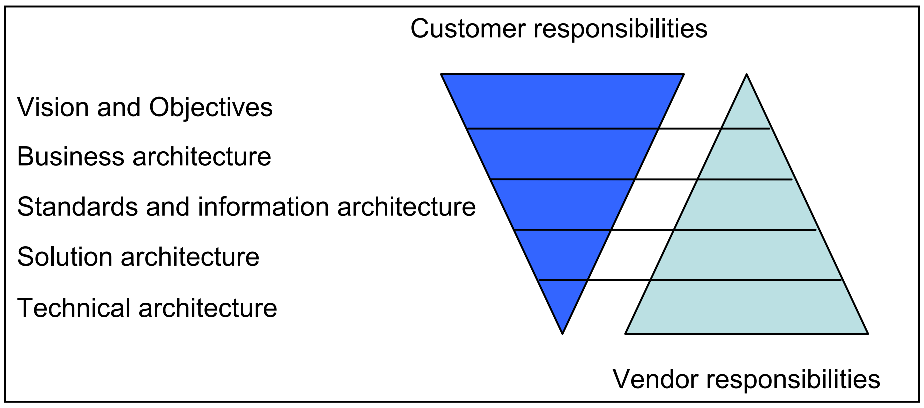


Figure 5.1 Balancing between customer's and vendor's responsibilities.

Some of the implications are:

- You as a customer should be able to manage and control not only the visions and objectives of your business, but also the business architecture, standards and information architecture.
- The suppliers will provide you with knowledge as well. To be able to make educated decisions, you should be in the driver's seat. Otherwise the vendors will determine the definitions and the possibilities of your world.
- This means being able to understand, define, decide, and describe your business in terms of processes and workflow.
- To ensure interoperability you must agree on some vital business objects and their definitions (e.g. what is a *programme*). This means that it cannot be left up to each individual department or person to choose how descriptions are made or used (even though some translation may be provided in the user-interface). You have to manage that.
- You must get the most important Metadata right at the beginning of your value-chain. Otherwise interoperability will be hard to achieve and rational behaviour will be replaced by subsequent re-work on the same data.
- As vertical systems dissolve into a diversified and distributed environment of components, aligned with your business processes, you must manage and control based on your processes. This means that, less and less often, a business will "own" and decide alone, how to manage and use a system.
- As a result of this you may need incentives from the processes and not from the line-organisation alone.
- You will benefit from having a cross-organisational board of technology advisers to ensure and co-ordinate strategic decisions across business units.
- You must migrate towards handling technology as convergent. Not separated into IT and the traditional broadcast technology.
- Whether you place the responsibility in an ICC or in the business units, or specify it as a common corporate role, business development will increasingly mean system- and IT-development. The same is happening in other business areas (e.g. banking, internet whole-sale, etc.).
- The drivers for business opportunities will, besides from good innovative content, increasingly come from your ability to make your systems work together. So you must make sure to have the right skills and processes in your business unit and to be able to see and value these new possibilities which are coming from technology.
- You will no longer be able to understand and control the solution architecture and, even less, the technical architecture. So let go, and concentrate on the business and information architecture, including internal standards and policies as well as strategic choices of technology platforms and standards.

6. Requirements

This is a basic set of requirements, taken from P/MDP Group members' experiences. It cannot of course be a complete match for your situation as, for example, the size of different broadcasters' operations leads to different requirements and priorities: one production team can easily have a common understanding of terminology used in a data model, but if your operation consists of 150 people, you may need a more formal sharing of information.

Special broadcast requirements
There are only two basic areas where broadcast presents special requirements compared to other users of IT technology:

1. Hard real-time requirements.
2. Very large file sizes, bandwidths, etc.

6.1 Interoperability / openness

- Interoperability and ease of integration are mandatory, systems will not be operated in isolation:
- Systems should provide an interoperability layer that allows communication between disparate parties (e.g. no closed middleware).
- The interoperability layer should follow open or de-facto standards, to prevent vendor lock-in and to maximise compatibility.
- Systems need to feature well-documented interfaces.
- Systems should provide well-documented semantics of the data(-model) used (otherwise you will be in a data lock-in situation).
- Interfaces should be programming language independent.
- Standard system interfaces, in terms of well-known methods and behaviours, are essential for reaching even the lowest levels of interoperability.
- Standard intermediate data models for the exchange are an a priori condition to allow semantic consistency through the whole integrated system.

6.2 Architecture

- Systems should excel in a particular functionality/task, while complying with, and leveraging, common/established infrastructure components as much as possible.
- Systems should be 'broken up' into smaller modules, to allow vendor independent replacements.
- Systems should expose rich functionality, made accessible through open standards.
- Relevant system components should support transactions, to guarantee predictable behaviour.
- Data and essence formats should be consistent throughout the whole solution.
- The underlying network should provide sufficient bandwidth for the system, including all AV transports.
- Service discovery should be possible using functional addressing.
- Systems should support common/established interaction patterns such as synchronous, asynchronous and real-time interaction; message broker-based and direct connections.
- There exist many content formats: systems should support 'handshaking' on them all.

6.3 Security

- Distributed user directories should be supported.
- Systems should use existing security and enterprise directory technologies.
- Systems should comply with and use common/established security and enterprise directory infrastructure components.

- Systems should support access control on material/information.
- Systems should protect privacy of user information.
- Systems should assure integrity of material/information.
- Systems should audit access to material/information.

6.4 System management/performance

- Systems need to provide detailed information on performance characteristics, especially on their scalability (e.g. how many clients will be able to work with the current set-up, data size limitations, etc.).
- Systems should allow for quality-of-service, e.g., prioritisation of traffic.
- Systems should facilitate monitoring of system components, data flows, etc.
- A common file transfer protocol should be supported (e.g. FTP).
- Slow performing systems must be handled appropriately.
- For offline systems, retries (resends) should be possible.
- Automatic discovery of services must be possible.
- It is desirable to manage the system with a central management system.

6.5 Real-time

- Real-time requirements should be fulfilled in a guaranteed way (e.g. setting up special paths for material transport).
- Specifications should distinguish between: hard real-time, soft real-time and non real-time.

6.6 Legacy system integration

- Systems should be able to adhere to/integrate with the broadcasters' data model (incl. business objects).

7. Standards

7.1 Reference integration model

Looking at the overall generalised structure for networked devices, this report also uses the reference integration model below (figure 1.1). This model clearly shows that common pervasive services are already in place and used in many other industries. These pervasive services are continually being developed by the technologies used in the IT world.

Examples of standards that exist in the physical network are listed in section 7.2.1 while pervasive service examples are listed in section 7.2.2. There are very specialised IT networks designed to solve specific service issues (such as MPLS and ATM) and it may well be appropriate for media production and broadcast standards bodies to leverage their special qualities to provide services to our industry as required (e.g. SMPTE 310M, AES47).

It is important to note that this layered view is shown purely from a standardisation perspective and does **not** reflect the technical dependencies of applications sitting in the broadcast domain area.

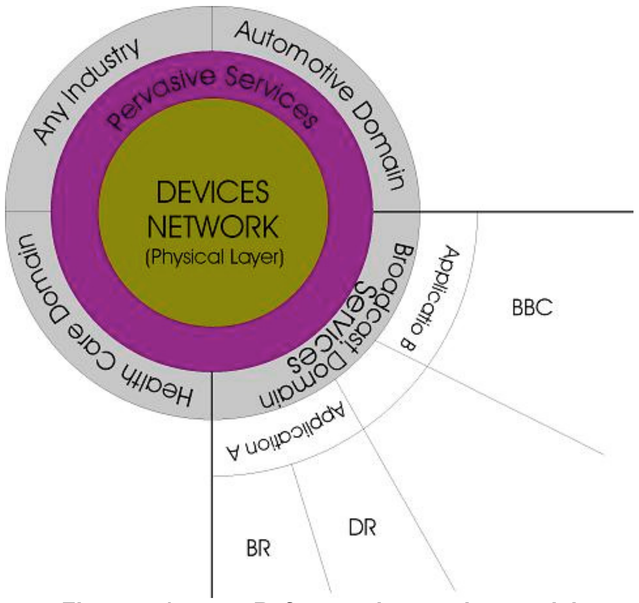


Figure 7.1 **Reference integration model**

Above the standard IT services sit the specialised domain services that are required to deal with particular media production and broadcast issues. It is in this area that this report attempts to explain how and why our industry should develop standardised structures. Section 7.2.3 lists relevant standards where they exist. However, there are a number of areas yet to be developed.

There are also business specific services which are only of relevance to a single organisation. These are clearly of little relevance for standardisation work as they are likely to be of use only to the original business process that designed them, though they might have some relevance for other organisations as well. Such areas are shown in figure 7.1 by means of the three broadcasters (BBC, BR and DR).

7.1.1 An example approach

The approach in this chapter is based on the ingest example analysed in this report. The overall use case for this example is shown in figure 7.2.

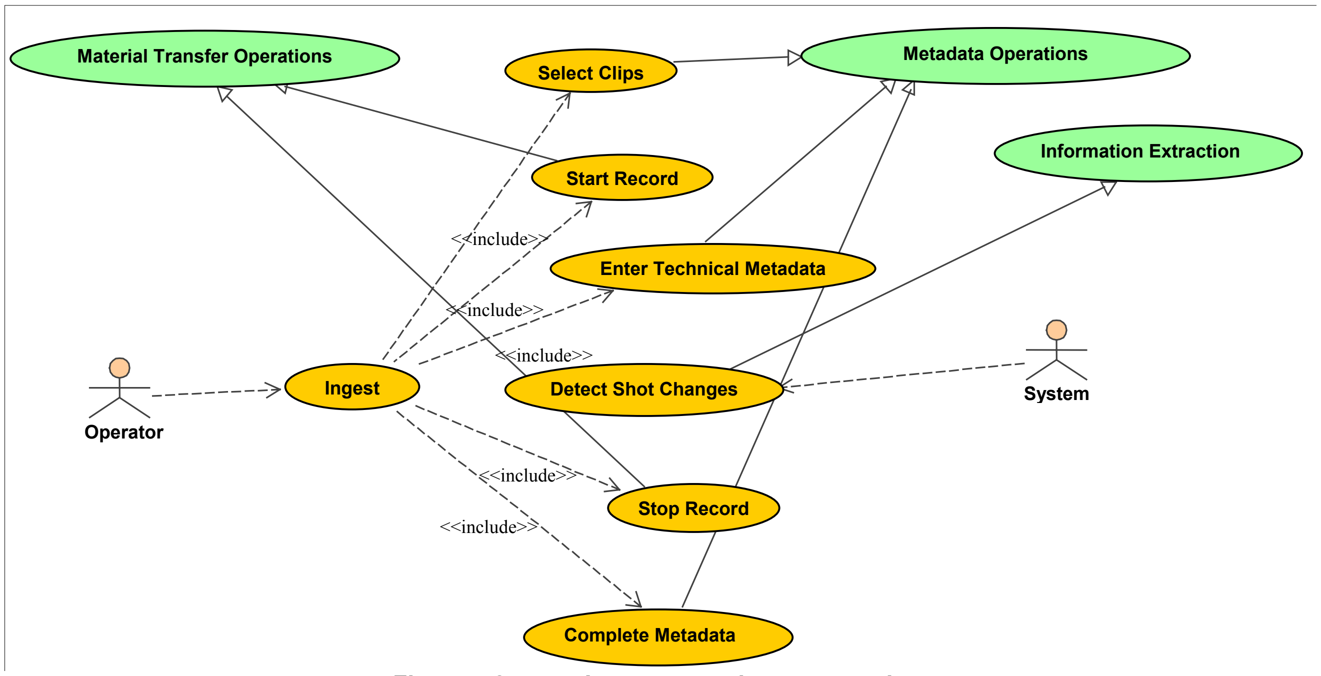


Figure 7.2 Ingest example use case view

Taking Figure 7. as a reference, for each of the identified areas, a further sub-classification is provided in the form of groupings of standards. Those specifications are deemed important in the context of the example. Of course, these lists should not be regarded as being exhaustive. Rather, they represent a compendium of the most relevant technologies which should be taken into account by any EBU member considering starting system integration activities related to the ingest use case.

7.1.2 Standards relevance determination

The role of the identified standards with regard to our ingest case is specified by:

- Associating the standard to the high level activities that are realised by the system.
- Identifying the main level of refinement at which the standard becomes important:
  - Logical integration: the standard provides logical interfaces realising some of the needed services. The standards falling in this category are relevant during the initial phases of the integration/design process.
  - Implementation: the standard provides components with which to realise the system. The standards falling in this category become relevant when the logical and functional structure of the system is determined.
  - Deployment: the standard provides facilities for the physical setting up of the system. The standards falling in this category become relevant when it is time for physically instantiating the system.

  Note that some standards or standards families may fall into more than one category
- Identifying the standards' components involved and their possible configuration parameters. Note that configurations may depend on the system's logical structure and user requirements. For example: quality-of-service considerations at the user level are reflected into the configuration parameters of the network components.

7.2 Identified relevant standards

7.2.1 Devices and networks

7.2.1.1 Broadcast devices connectivity

Audio/video standards

- SDI, Serial Digital Interface, SMPTE 259M

- SDTI, Serial Data Transport Interface, SMPTE 305.2M
- Time and control Code, SMPTE 12M
- AES/EBU digital audio interface, AES3-2003, EBU Tech 3250-E
- Transmission of digital audio over ATM networks, AES47-2002, IEC 62365

Associated activities: Ingest.Material Transfer Operations

Refinement: implementation

<u>Control</u>

- RS232 serial interface
- RS422 serial interface
- GPI, General Peripheral Interface
- SNMP, Simple Network Management Protocol, RFC 1157 (IETF)
- Common Control Interface for digital audio and video products on ATM networks, IEC 62379

Associated activities: Ingest.Start Record, Ingest.Stop Record

Refinement: implementation, deployment

7.2.1.2 Information Technology connectivity

<u>Network protocols & services</u>

Figure 7.3 depicts an overview of widely used network standards.

Other relevant standards are:

- SNMP, Simple Network Management Protocol, RFC 1157 (IETF)
- Common Control Interface for digital audio and video products on (ATM) networks, IEC 62379
- IIOP, Internet Inter-ORB Protocol
- RPC, Remote Procedure Call (at the session layer)
- RJxx (at the physical layer)

Associated activities: Ingest

Refinement: application - presentation standards: logical integration
transport - session standards: implementation
network - data link - physical standards: deployment

<u>Databases management and access</u>

- ODBC, Open DataBase Connectivity
- JDBC, Java DataBase Connectivity
- SQL, Structured Query Language

Associated activities: Ingest.Metadata Operations, Ingest.Information Extraction

Refinement: implementation

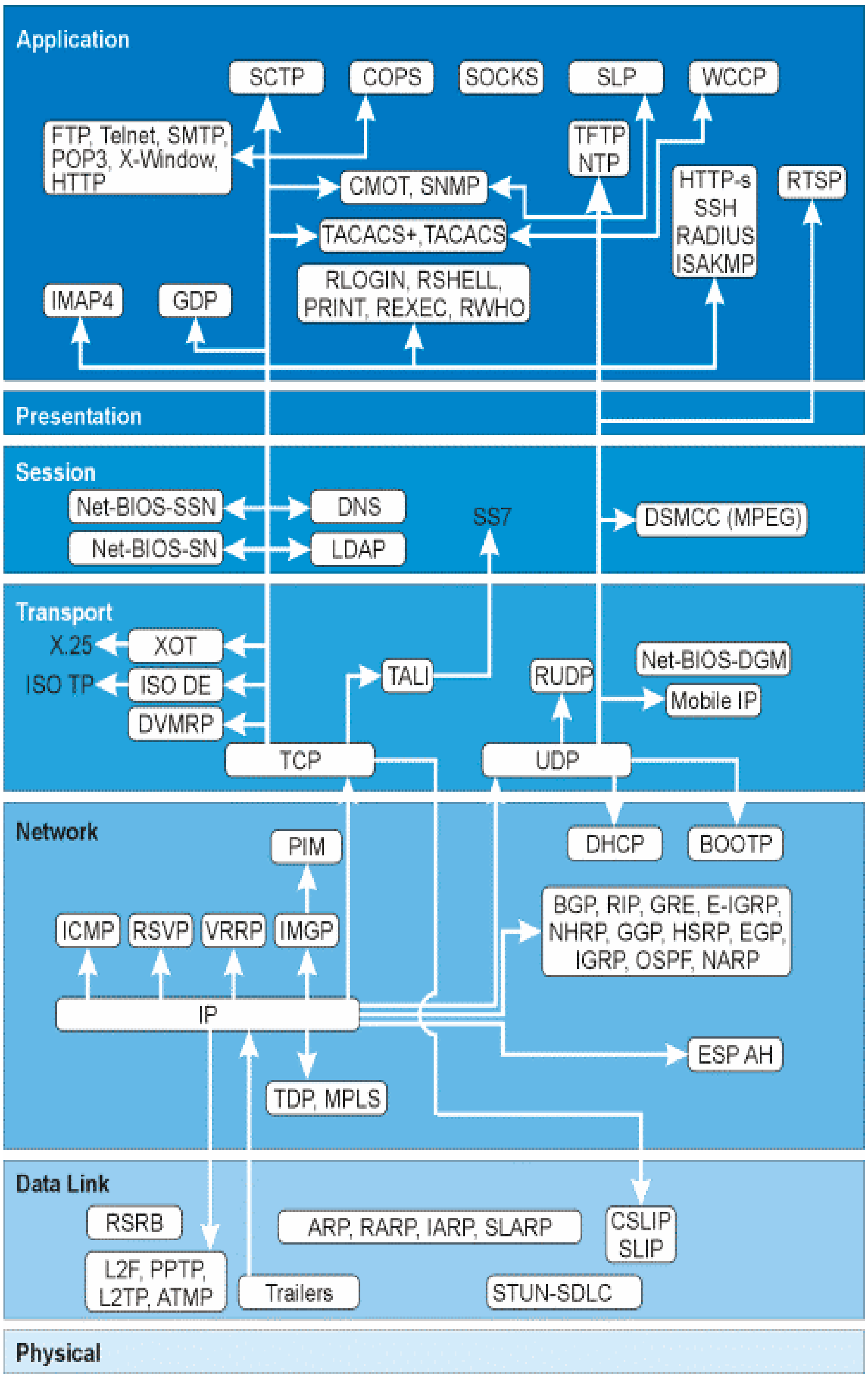


Figure 7.3 **The TCP/IP protocol hierarchy** ***(source: www.protocols.com)***

<u>I/O computer architectures</u>

- RS232, serial interface
- PS/2 (IBM)
- SCSI, Small Computer System Interface
- S-ATA, Serial - ATA
- USB, Universal Serial Bus
- PCI-X, PCI eXtended

Associated activities:	Ingest
Refinement:	deployment

7.2.2 Pervasive Services

7.2.2.1 Message Transport

- XML, eXtensible Markup Language
- SOAP, Simple Object Access Protocol
- JMS, Java Messaging Service

Associated activities:	Ingest
Refinement:	implementation

7.2.2.2 Time Reference

- NTP, Network Time Protocol

Associated activities:	Ingest.Material Transfer Operations
Refinement:	implementation

7.2.2.3 Transaction

<u>Workflow management</u>

- WFMC, WorkFlow Management Coalition standards

Associated activities:	Ingest
Refinement:	logical integration

<u>Transaction management</u>

- XAML, eXtensible Application Markup Language

Associated activities:	Ingest.Metadata Operations
Refinement:	logical Integration

7.2.2.4 Authentication

- LDAP, Lightweight Directory Access Protocol
- EAP, Extensible Authentication Protocol
- TLS, Transport Layer Security Protocol

Associated activities:	Ingest
Refinement:	implementation

7.2.2.5 Transversal

Web services related standards

- WSDL, Web Service Definition Language
- UDDI, Universal Description Discovery and Integration

Associated activities:	Ingest
Refinement:	implementation

CORBA

- CORBAServices

Associated activities:	Ingest
Refinement:	implementation

7.2.3 Broadcast Domain Services

7.2.3.1 Broadcast foundation services

These are broadcast specific services that are useful to most of the broadcast applications, the "bread and butter" services. For these there is lack of standardisation. Through the identification of common broadcast and production services in appropriate bodies, standards could be developed for our industry. These must include interface components for accessing these services.

7.2.3.2 Encoding standards

- MPEG-2, ISO/IEC 13818-2, Information technology -- Generic coding of moving pictures and associated audio information: Video
- MPEG-4
- H264
- JPEG
- DV

Associated activities:	Ingest.Material Transfer Operations
Refinement:	logical integration

7.2.3.3 Essence file & wrapping formats

- AAF, Advanced Authoring Format (AAF Association)
- ASF, Advanced Systems Format (Microsoft)
- AVI, Audio Video Interleaved (Microsoft)
- BWF, Broadcast Wave Format (EBU Tech 3285)
- DV-DIF
- MXF, Material Exchange Format, SMPTE 377M

Associated activities:	Ingest.Material Transfer Operations, Ingest.Metadata Operations
Refinement:	logical integration

7.2.3.4 Information models

Domain models

- IFLA model
- MPEG-7

- MADL (RAI proprietary)
- SMEF (BBC proprietary)

Associated activities:	Ingest.Metadata Operations
Refinement:	logical integration

<u>Metadata dictionaries</u>

- SMPTE RP205
- P_META, EBU Tech 3295
- Dublin Core, EBU Tech 3293
- DMS-1, MXF Descriptive Metadata Scheme - 1, SMPTE 380M

Associated activities:	Ingest.Metadata Operations
Refinement:	logical integration

7.2.4 Transversal standards

7.2.4.1 Multimedia

- DirectShow platform (Microsoft)
- SMIL, Synchronised Multimedia Integration Language (W3C)
- SVG, Scalable Vector Graphics (W3C)
- ASX (Microsoft)

Associated activities:	Ingest.Information Extraction
Refinement:	logical integration

7.2.4.2 Operating systems

- Open Operating Systems (GNU/Linux)
- POSIX, ISO/IEC 9945

Associated activities:	Ingest
Refinement:	implementation

7.2.4.3 Information Exchange & Representation

- XML, eXtensible Markup Language (W3C)
- DOM, Document Object Model (W3C)
- XPath language (W3C)
- XSLT, eXtensible Stylesheet Language Transformations (W3C)
- XQuery, an XML query language (W3C)
- KLV, SMPTE 336M

Associated activities:	Ingest.Metadata Operations
Refinement:	implementation

7.2.4.4 Advanced knowledge representation

- RDF, Resource Description Framework
- RDFS, Resource Description Framework Schema
- DAML+OIL, DARPA Agent Markup Language + Ontology Inference Layer

Associated activities:	Ingest.Metadata Operations
Refinement:	logical integration

7.3 Utility standards

7.3.1 System interfaces specification

- IDL, Interface Definition Language

7.3.2 Design and modelling facilities

7.3.2.1 Design Patterns

- XMI, XML Metadata Interchange

7.3.2.2 Modelling languages and frameworks

- UML, Unified Modelling Language
- MOF, Managed Object Format
- CWM, Common Warehouse Metamodel (CWM Forum)
- OCL, Object Constraint Language
- XML Schema
- ERD, Entity Relationship Diagrams

7.3.3 Cross-platform Programming languages

- Java
- PHP
- Perl
- C++

7.3.4 Standardisation bodies & consortiums

- AES, Audio Engineering Society
- ANSI, American National Standards Institute
- IEC, International Electrotechnical Commission
- IEE, Institution of Electrical Engineers
- IETF, Internet Engineering Task Force
- ISO, International Organisation for Standardisation
- ITU, International Telecommunication Union
- OASIS, Organisation for the Advancement of Structured Information
- OMG, Object Management Group
- SMPTE, Society of Motion Pictures and Television Engineers
- W3C, World Wide Web Consortium

7.4 Conclusions

Business objects are an important issue and need to be considered for standardisation processes, if generally relevant (some business objects may be specific to specific organisations).

Within the Broadcast Domain services, industry wide common services should be developed along with the standardisation of process / component interfaces in commonly used services. These services and interfaces can be developed within current industry bodies as well as by new industry forums if necessary.

See chapter 8 for more information on recommended future work.

8. Recommendations for future work

(Valid at the time of release of the P/MDP document)

8.1 Future work items

8.1.1 Inside EBU

1. Stress (to P/MAG) the urgency of specifying the core elements of a Metadata model

As the vendors session clearly indicated an urgent need for having a core Metadata model as a basis from which to work, this should lead to action within EBU. At the very least, a study must be made as to whether a solution is deemed achievable. Focus should be on practical solutions, starting with agreement on some common business objects.

2. Consider building on the work of the N/CNCS Project Group

The N/CNCS Project Group is looking at control of live production structures. Currently, as equipment manufacturers develop network interfaces for their equipment, they tend to also develop a proprietary API (***a**pplication **p**rogramming **i**nterface*) to control and monitor that product through the network. Within sizeable production and broadcast organisations, this could (and in some cases is already) leading to the requirement to manage a very large number of APIs, each of which requires specialist knowledge of the products concerned. This in turn makes managing live production structure a complex and expensive issue with many potential points where errors and failures can occur.

The N/CNCS project group (Common Network Command & Control Strategy) will identify the common control and monitoring requirements of live broadcast structure and develop recommendations for members, along with ensuring that these are integrated with similar standardisation work being carried out in the IEC.

3a. Consider forming a Project Group on super-services

This Group should identify common super-services amongst Members. It does not have to define data-models in detail (see 3b).

3b. Develop an OMG broadcast domain model

Having a domain model is a requirement. However, the organisation that should develop it is open to discussion. OMG is certainly a good fit, and there is a suggestion that involving our industry in that work would be a good start (several manufacturers are already heavily involved in the OMG, but no-one from the broadcast environment). Another option might be to have SMPTE start such an initiative, or, even better maybe, have the two work together?

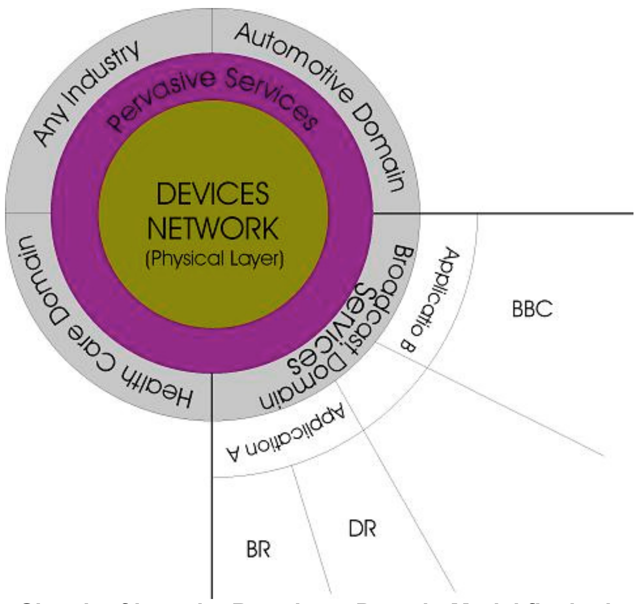


Figure 8.1 **Sketch of how the Broadcast Domain Model fits in the larger IT world.**

4. Form a Training Group on IT skills

This Group should try to narrow the skills gap between broadcast & IT staff by making sure that information (knowledge) goes both ways. The EBU Training Department is advised to set up a group, investigating the requirements for the training skills needed for modelling broadcast infrastructures. Its courses would allow Members to be educated in modelling skills (including UML). Courses like these are already available, but are not tailored to media/broadcast. Another problem is how to 'spread the word' so that the right people are made aware.

5. Form / Consider cross-EBU workshops on new kinds of management in a new broadcast world

As the technical environment is changing, the management and control changes as well. Not only due to the tighter integration of business decisions and technology decisions, but also with regard to the question on how to manage investments appropriately. Some of the mechanisms commonly used in the IT-world and in larger complex organisations might be of help here. But appropriate adjustments for the broadcast world should be made. Sharing experiences, visions of threats and solutions should be carried out in EBU forums and involve both the content and the technical aspects.

6. Later, form a Project Group collecting best practices and identifying patterns

This Group should derive best practices and identify patterns in real-world solutions, developed on the basis on the Domain Model (as in 3b).

Note that efforts similar to 6 are currently being performed in other industries, but cannot be started without a domain model (see 3b) being available. Thus, 6 is dependent on 3b.

8.1.2 Outside EBU

1. Provide the results of the P/MDP work to SMPTE

This should include the identification of missing standardisation related to Middleware and the opinion/approach derived from the vendors' meetings. The goal should be to obtain feedback on the conclusions (agree, disagree, different viewpoints) and learn where consensus is available. Also any future (joint) standardisation work should be planned. Note that middleware in general can be regarded as a logical follow-up of the EBU/SMPTE Task Force work [5].

2. Publish this report publicly

Public availability would help some of the issues raised to be taken forward with vendors and outside bodies.

8.2 Standardisation activities

In our discussions with vendors on standardising middleware and related technology, it was clear that:

- Many standards already exist.
- Most come from the IT industry.
- Standardising a single middleware is undesirable (and probably unachievable).
- Standards are a good help for documenting systems (a time-consuming task).

There are two areas where standardisation efforts would be welcome:

8.2.1 A common (core) Metadata model

There is a lack of a common (core) Metadata model. This is the priority one issue for most vendors. Existing Metadata specifications have not been very successful and are seen as too large/vague to apply. There is a strong demand for a more modular and business oriented approach: e.g. a simple subset with a coherent structure. This must include the semantics (= unambiguous meaning) of the elements it provides.

The situation is urgent and it seems EBU could play a strong role here. This correlates with similar conclusions from other Groups/events (e.g. P/TVFile, MXF Workshop, etc.).

8.2.2 'Bread & butter' services

The business services commonly used by broadcasters, such as ingest, play-out, scheduling, are the broadcasters' 'bread & butter' services. It seems that standardisation of middleware/system integration architectures could best take place at this level.

These primary services could all be broken down into non-competitive and common technical services (play, record, transfer, transcode, etc.) and all have a need to interface.

Besides services, relevant business objects should be specified as well. The specifications should not dictate technology but allow for competition. The effort should start with words & models. Once the services are defined, it should be investigated whether toolkits, APIs, etc., would be of value. Where standards are immature or ambiguous, reference implementations would be an advantage.

Who should standardise these services?

Various vendors noted that the (broadcast) standard bodies tend to be dominated by large vendors and also, participants are typically engineers, often detached from business practice. Further, the broadcast standardisation process is perceived as slow.

IT standardisation is seen (at least for some organisations) as faster. This also means participation may be more intensive. IT standardisation bodies are seen as key in the middleware area.

The conclusion is that a combination of an IT body (e.g. OMG) working together with a broadcast group (e.g. EBU, SMPTE) could be a viable route. There are also standardisation bodies such as the IEC who are useful as they work in both the media and IT fields. An alternative could be to try to start an interoperability forum, as there currently is no industry forum where vendors & customers can work collaboratively before being in a sales situation.

Finally, it was pointed out that there is a cost issue for smaller companies: any process should be resource-efficient (inexpensive), for example by making use of the Internet, instead of having physical meetings.

When should standardisation start?

- It seems that currently some manufacturers see little urgency for standardising middleware related topics. Many of them have just started to implement file-based technologies, such as AAF/MXF support in their products, and they need time to achieve sufficient Return On Investment.

- Middleware elements have already been around for more than ten years and, as one manufacturer put it: 'there is no immediate penalty if you don't use them'. However, although the middleware that has been in use may offer technical solutions, the semantics for the broadcast & production industry are still in their infancy.

- Most vendors have no major incentive to standardise interfaces as long as broadcasters cannot agree on major definitions of business objects (what is a programme, what is a service, etc.). They will be happy to deliver all the different flavours of the same and all the different kind of glue or kit needed to make it fit together. They will hardly be the driver of a broadcast domain model.

But-

- In view of the developing convergence of traditional broadcasting techniques with IT techniques, and also that any development today has a strong business element, broadcasters now need the ability to specify IT based applications and business structures within their organisations. It is now a matter of urgency that standardisation work begins, to enable users to specify functionality between products.

- Taking into account that standardisation takes time and the use of middleware is clearly a common factor in other (influencing) industries, the need to start as soon as possible seems the inescapable.

8.3 Educational

- To provide a tutorial Workshop/Seminar in 2005.

- EBU should provide a management tutorial to provide guidance on how to manage the system integration process, set-up an ICC, etc.

  - The EBU should provide a repository for super-services, semantic descriptions, reference models, etc. This could be in the form of links to other repositories.

9. Golden rules

This chapter provides 20 'Golden Rules' regarding system integration, derived from Members' experience in system integration projects.

9.1 Your business

#	Advice	Punch line	Details or constraints	Arguments
1	Don't talk of systems, talk of services or functions	Systems are dissolving	In a integrated n-layer architecture it makes less and less sense to take the system approach	Many decisions and designs are misleading or misunderstood because they only look at systems.
2	Look at middleware as a vital part of your infrastructure	Middleware *is* infrastructure	Like IP-networks, servers, storage etc. middleware is an integrated part of the infrastructure and should be treated as such - financially as well as with regard to management and decisions	No single project or part of your organisation is able to carry the whole burden of infrastructure. I.e. to establish or maintain middleware platforms, adapters etc. The development of adapters could be included in the different local projects
3	Change from system ownership to process ownership	Align with processes	Complex decision making is possible in a mix of traditionally vertical decision systems and horizontal systems like system ownership or process ownership	In an integrated environment no single department or director has control over the investment, acquisition, maintenance or development of large systems and their components as they are used in a widespread manner
4	Be clear about the information of your world	Know your world	Have a universal central repository so you can find things. Implement a structured information repository of your business processes, your workflow, the systems, the data model, the physical deployment and the relations between all of them	You might end up in similar complex chaos like you had with many 1:1 integrations - just in a new and highly integrated way on a new platform
5	Define the responsibilities related to a central repository of information	Know who manages your information	Different roles to change information have to be defined. In the business layer as well as in the system area. Change management routines have to be implemented.	If you don't define who decide and change information, you'll end up in information chaos.
6	Process descriptions should include business rules	Define logical steps of decisions	The business rules are the point were processes and work-flows meet functionality in the systems.	Working with flow-charts you tend to miss the business rules. If business rules not described with use-cases it is hard to decide if they should be carried out by human interaction or automated by systems.
7	Define your business objects and data model	Own your world	All vital parts of your business definition have to be under your control.	If not, you must adapt to the vendors definition of the world and how to act using their business and data model. Also data lock-in is a danger if you don't own your model!
8	Have clear definitions of all the process workflows in your organisation	Look at middleware from the business perspective	Descriptions of processes and workflows are as important as descriptions of systems and interfaces. You can either describe processes and workflows or describe use-cases.	You need to understand how you work to be able to design the systems. The consequences of changes at either end should be traceable at the other end.
9	Aim for clear and complete resource management across all systems	Have full resource management	You need to know what you are using and how much it is costing at any stage of your work	Note: this is much more complex in a horizontal environment which stresses the need for administration and management. But is this related to middleware or just a general rule?? The feeling is that it is needed. But the implications is not clear
10	Aim for unlimited network bandwidth, available everywhere so that you can make programmes without restriction and always get things when you need them	We want fast access	In a horizontal n-layer structure more information is transferred and more often. Therefore it is predicted that scalability, segmentation and Quality of Service are strongly needed.	If the network doesn't follow the requirements set by the integration architecture, poorer performance than today can be expected.

9.2 Vendors

#	Advice	Punch line	Details or constraints	Arguments
11	Break down your system into its main functional parts	We want open modularity	For each vendor, major functional blocks have to be separable inside the system framework. Don’t break it down in too-detailed modules. We want your expertise in combining low level functionality into good work-flows	We will never buy their entire packet for a whole broadcast station. We would love to buy their playout system, their high-level planning system, their mid-range editing tool or their archive - but never in a proprietary combined solution
12	Functional parts with clear borderlines shall be accessible with open standards	We want open standards	An architecture that supports SOA. We want to be able to use services inside each functional part. The services shall be available through open technology standards (COM, DCOM, JCA etc).	Open modular systems with proprietary APIs are not good enough
13	Manufacture interoperable and exchangeable sub-systems - no matter what	Integration is mandatory	Regardless of the level of modularity and implementation of a horizontal n-tier concept a integration layer is required in your system	We are not able to handle systems as isolated islands in our business and we know were integration is needed
14	We require completely transparent low latency structure for uncompressed live programme streams	Real-time systems are an important part of our business	A clear plan for handling real-time material should be made: How are time-critical requests and responses handled, how is this handling connected to the actual live-streaming or file-transfer	If you don’t have a plan for this, the vendors might not pay any attention. Or you might end up with a collection of quick-fixes as each time there is a problem, it is patched.
15	Apply or integrate to our business objects and data model	We own our world in partnership with you	We hold the responsibility to have vital parts of our business definition under our control. It is vital that you either adapt or integrate to that with open interfaces. We will be willing to adapt parts of your business or data model if you can convince us about the benefit in our world.	If not, you bear the risk that implementation or use in our complex environment will fail even if your system is great.
16	We require universal media ingest points available anywhere on the structure in any numbers	Ingest is infrastructure	The ingest of material is a widespread and long-term facility which has to be independent of the systems for editing, play-out etc. It is strongly related to the archives and the handling of your own Metadata and data model.	If not, you might end op with restrictions to the use of your Metadata and data model, as the ingest is tightly connected to a single system solution.
17	Prepare for the future by preparing to expose more functionality than currently required.	Have the ability to manage what you expose for future use	Technology support and a lot of well-defined services shall be required even though we do not need them when we buy (or build) our system.	As with environment and pollution a safety-first approach should be adopted in this area. Openness can sometimes be very short-sighted.
18	We want transaction support	Keep system and data information consistent	Transaction support is needed in the systems that connect in the integration layer. The transaction support should guarantee that either the transaction updates the targeting system or the information is not altered at all. The information in the target system should always be in a consistent state. If long transactions are used (that span several weeks) it must be possible to restore the information with compensation transactions.	Even though this is what a broker does, it is important that the different products allow this.
19	We need security	Security is essential	In the middleware there must be support for identification, authentication and integrity across systems and data sources. Traditional system rights management is not sufficient when dealing with data carrying system integration	The systems must know who are using them, what permissions they have, that they are who they claim, that information is not changed unless permitted etc.
20	Manufacture interoperable and exchangeable storage	Free choice of storage	The system shall be able to use the strategic choice of your company regarding databases and storage (given they deliver the appropriate performance)	If not, it is hard to make a common interface to the data layer

Appendices

To be read by specialists or people with special interest in the subject

Annex A BR project - detailed description

A.1 System overview

The system can be divided into three layers, as shown in figure A.1.

1. Storage layer
2. Transport layer
3. Data management layer

A.2 Storage layer

The bottom *storage-layer* is responsible for storing and managing all kinds of essence on file-based data carriers (online and near-online). The main systems in this layer consist of an *archive-storage* with disk-buffer, two tape libraries and the DIVArchive management software, responsible for managing at least 8500hrs archived files[2] per year.

For each object in the archive-storage a low-resolution browsing copy in 1.5 Mbit/s MPEG-1 quality is stored in the *browsing-storage.* This is a two level disc cache applying SAN[3]-topology and which is exclusively controlled by the *content-management-system* in the data-management-layer.

For the AVID non-linear editing applications there is an AVID-UNITY system as *editing-storage* platform supplied. This system has a total of five independent components, each with approx. 100hrs capacity (D10-IMX).

For programme playout, separate *transmission-storage* will be used - consisting of two independent video-servers with approx. 200 hrs capacity (IMX 50 Mbit/s). The management of the Content stored is carried out by the Harris *transmission-automation* system in the *data-management-layer*.

For tape, and feed-ingesting, as well as for news play-out, a huge[4] *central-storage* system forms the core of BR´s production environment. It also provides material to the *transmission-server* and buffers all material to be archived. The central storage consists of three independent nodes, connected via Gigabit-Ethernet. Each node is based also on a SAN topology and supports at least 30 parallel SDI I/O's.

Exchange of essence[5] between the different systems within the lower layer is via Gigabit Ethernet (ftp). The Harris extended VDCP[6] (*video-server real time control*) protocol is used to control the storage-layer from the transport-layer.

A.3 Transport layer

The next level is the *transport-layer*[7]. This layer manages the transfers between the single systems within the storage layer. Transport will mainly consist of *ftp*-transfers, but real time transfers via base-band video (SDI) is also facilitated. Triggering of the transfers is a responsibility of the *data-management-layer*.

To allow interaction with the *storage-layer,* the *transport-layer* has to always know the actual inventory of files in the storage-systems. This is facilitated by HARRIS proprietary device-drivers, which operate at a very low abstraction[8] level for each storage-system. For initiating and controlling essence-transfers via *ftp* between the storage-systems, the *transport-layer* works as an *ftp-proxy-client*, using standard *ftp-commands*. For real time control, e.g. the control of video server ports inside the storage-layer, *VDCP* is used.

A special case is the control of the Archive-Storage. For this a special VACP[9] network protocol is implemented. HARRIS only sends archive or restore requests to the archive-system, while the control of the *ftp-transfers* is done by the DIV-Archive-software in the *storage-layer* itself.

[2] BR will archive files with Sony D10-IMX compression (MPEG-2-50 Mbit/s)
[3] SAN: Storage Area Network
[4] Storage capacity about 600 hrs (IMX 50 Mbit/s) per node.
[5] IMX inside an MXF-file structure – no descriptive Metadata is mapped.
[6] VDCP: Video Disc Control Protocol (industry standard from Harris)
[7] Global Media Transfer functionality and device-drivers inside ADC-100 system from Harris
[8] Receiving information directly from the file-system
[9] VACP: Video Archive Control Protocol (industry standard from Harris)

Communication with the top layer is implemented in two ways. The first is via a proprietary real-time network protocol for most of the HARRIS applications. Non-HARRIS applications inside the *data-management-layer* can interface to the *transport-layer* via IMS[10]. Thus, external systems can be used to place transfer requests.

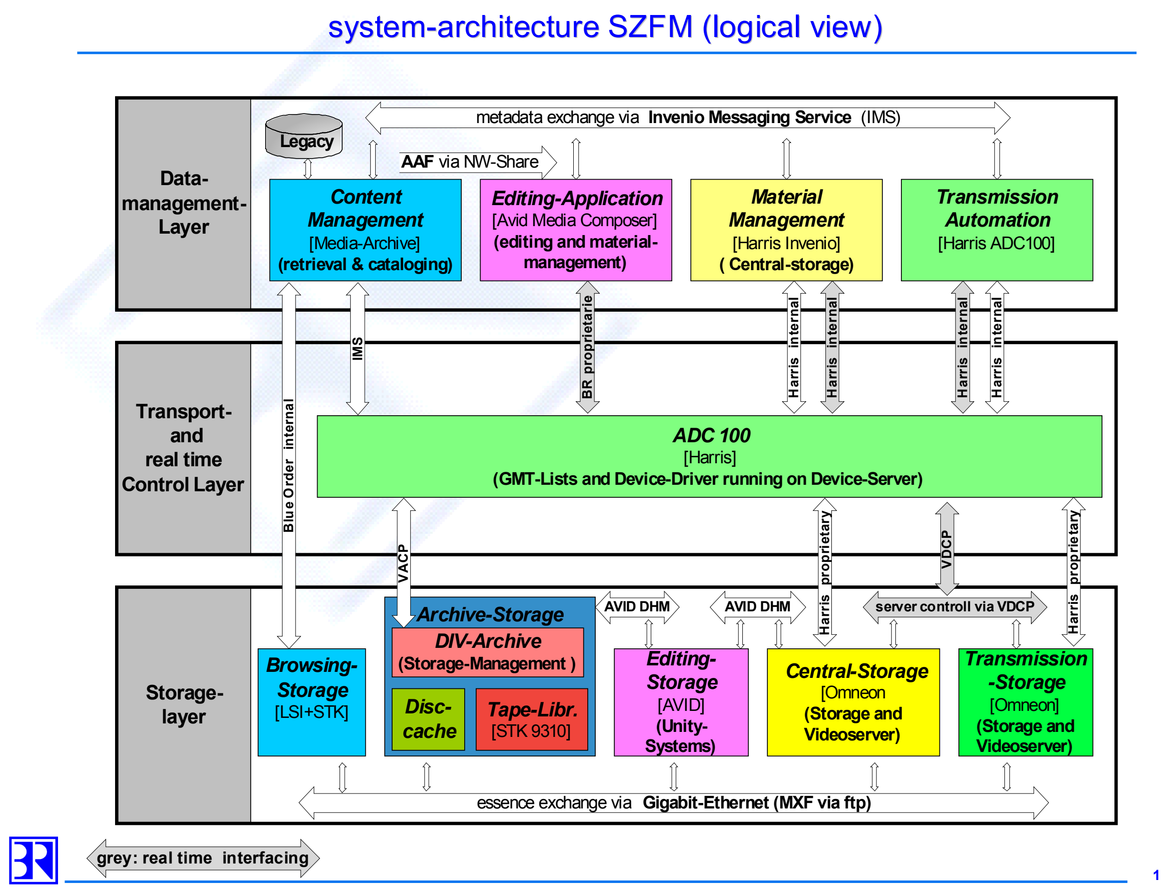

system-architecture SZFM (logical view)
Data-management-Layer
Legacy
metadata exchange via Invenio Messaging Service (IMS)
AAF via NW-Share
Content Management [Media-Archive] (retrieval & cataloging)
Editing-Application [Avid Media Composer] (editing and material-management)
Material Management [Harris Invenio] (Central-storage)
Transmission Automation [Harris ADC100]
Blue Order internal
IMS
BR proprietarie
Harris internal
Transport- and real time Control Layer
ADC 100 [Harris] (GMT-Lists and Device-Driver running on Device-Server)
VACP
Harris proprietary
VDCP
Storage-layer
Browsing-Storage [LSI+STK]
Archive-Storage
DIV-Archive (Storage-Management)
Disc-cache
Tape-Libr. [STK 9310]
AVID DHM
Editing-Storage [AVID] (Unity-Systems)
Central-Storage [Omneon] (Storage and Videoserver)
server controll via VDCP
Transmission-Storage [Omneon] (Storage and Videoserver)
essence exchange via Gigabit-Ethernet (MXF via ftp)
grey: real time interfacing
1

Figure A.1 Overview of the BR project's system architecture

A.4 Data management layer

The top layer is the *data-management-layer*, which is responsible for all „user Metadata“ handling and for all interaction with the transport-layer (placing transfer-orders). Also, the data-management-layer is the man-machine interface with many kinds of user-applications. The interface between the components within the data-management-layer occurs via IMS. The only exception applies to the AVID *editing-applications,* which are interfaced from the CMS by providing processed AAF-files as a result of restore requests via network-shares (manual import).

Inside the *data-management-layer* layer a *Content Management System* (*Media-Archive* from *tecmath*) resides. This is a distributed, service-based system, used primarily for archivists and producers. It controls the generation and storage of browsing material. The most important functions are:

- retrieval and selection of feed- and archive-material (Browsing-Client),
- the ordering of material and the provision of tools for cataloguing of material that was stored in the archive-system.

The interfacing of legacy-systems (e.g. FESAD) is implemented via application-servers.

Further systems include the *Editing-Applications* from AVID (Media Composer) for all the editing tasks. Material management is accomplished by the Unity Media-Manager software. The applications have their own proprietary interface to the *editing-storage* in the storage-layer.

[13] IMS (Invenio Messaging Service): enables exchanging of XML-documents based on WEB-Services

Imagine, you have to manage 50.000 files via a „DOS Shell" - this is impossible! Therefore, a separate *Material-Management* application (HARRIS Invenio) is managing the files from the central storage using a „Windows Explorer" like user interface.

The Transmission-automation processes and provides material for transmission on the basis of transmission lists. It checks the transmission storage for missing material. Missing material is demanded from the storage-layer via the transmission-layer.

Annex B DR project - detailed description

As part of a larger corporate strategic development project the technological impact of the business strategic objectives was analysed in 1999 and 2000. One conclusion was the need to move from closed vertical systems into horizontal, n-layered, system architecture.

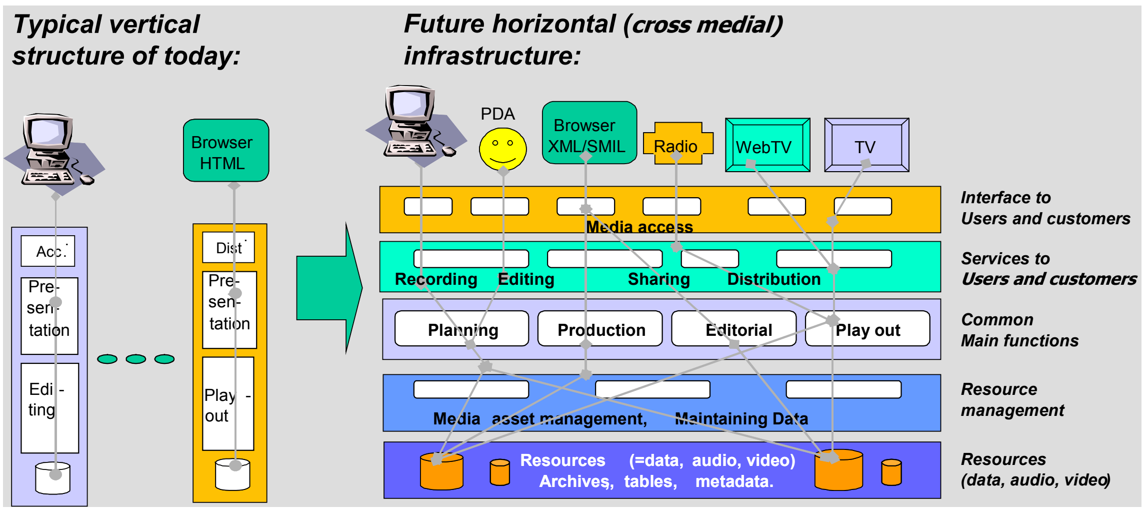

Typical vertical structure of today:
Future horizontal (cross medial) infrastructure:
Browser HTML
Acc.
Pre-sen-tation
Edi-ting
Dist
Pre-sen-tation
Play-out
PDA
Browser XML/SMIL
Radio
WebTV
TV
Media access
Recording
Editing
Sharing
Distribution
Planning
Production
Editorial
Play out
Media asset management, Maintaining Data
Resources (=data, audio, video) Archives, tables, metadata.
Interface to Users and customers
Services to Users and customers
Common Main functions
Resource management
Resources (data, audio, video)

Figure B.1 The migration from vertical systems to a n-layer approach

Research in available technologies, market trends and offers as well as structured analysis of the portfolio of systems in DR was carried out late 2001 and 2002. The different domains (Administration and Management, Research and Planning, Production, Play out and Distribution) was analysed, revealing the need for further integration in each of the three layers: Presentation, Application (logic) and Data. +60.

Major opportunities for better system integration were described, and a handful of the most important was chosen. 16 of them were picked out for 'proof of concept'. This POC should ensure the functionality of the chosen technologies and that the benefits and results expected would be fulfilled.

In particular, the need for working across DR's two strategic platforms - MS and Oracle - was investigated in the POCS.

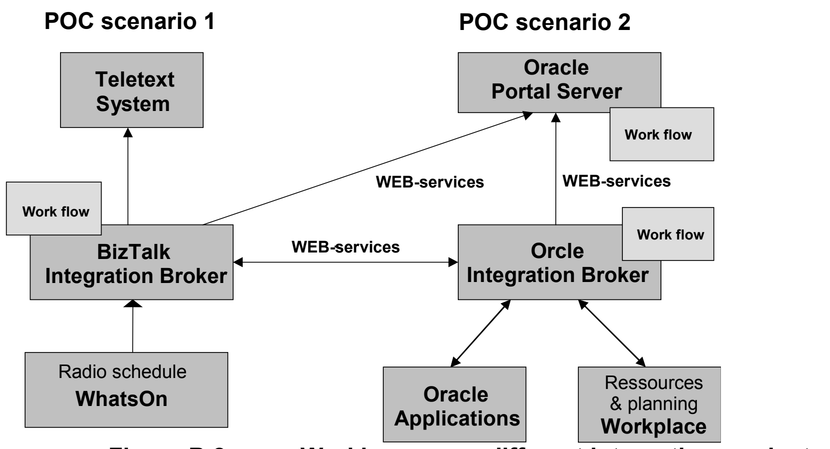


Figure B.2 Working across different integration products

- Over 80 requirements for an integration platform were specified.
- 8 standard requirements to all system vendors regarding system integration were set out.
- A set of DR standards and policies for system integration was developed.
- The Integration platform is acquired after market survey regarding performance and existing DR IT environment.

The objective was indicated as going towards a ‘hub and spoke’ architecture:

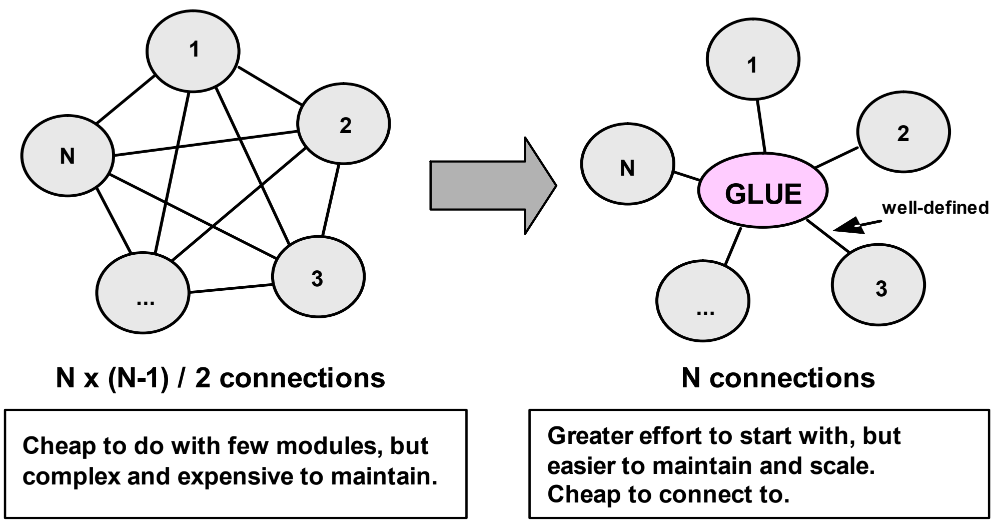

1
2
N
3
...
N x (N-1) / 2 connections
Cheap to do with few modules, but complex and expensive to maintain.
1
2
N
GLUE
well-defined
3
...
N connections
Greater effort to start with, but easier to maintain and scale. Cheap to connect to.

Figure B.3 From a spider's network to a ‘hub and spoke’ approach

The broker, as well as major parts of the policies, was in operation in spring 2003. A revised version of the standards and policies is underway based on the actual implementation and the first experiences. Particularly the strengthened focus on Service Oriented Architecture, which was just arriving as a challenger to broker based integration architecture in the early start of this project, is now taken into account.

B.1 System overview

As the different technologies, methods and protocols were examined, it turned out that no single layer or component of middleware could be defined between the logical layers: Presentation, Application (logic) and Data. It would require distorting the real world into a simplistic and symbolic understanding which could lead to trouble and confusion when making crucial decisions.

This was due to the fact that different dimensions of restrictions are present in the real world:

> We do have to deal with a variety: from not-layered systems across thick clients with rather large amount of logic in them to proper, layered applications with well-defined borders between the three (or more) layers.
>
> For each of these situations there might be different degrees of open and standardised interfaces to the different layers in an application. As an example, a perfect n-layered application can be totally closed and inaccessible through open and non-proprietary interfaces.

As a result, our concept system integration architecture looks rather complex, but at least everyone involved can agree on it and accept it as describing their world. It is illustrated on the next page.

The connection between each element in the concept model is described with preferred methods and technologies to use (Soap, XML, RMI, COM+ etc.).

The impact of this is that, although the concept on a high level can be adopted by all broadcasters, the more detailed view will look rather different as it will depend on the actual IT environment (.net, Java, Oracle, IBM, MS, BEA etc. and the different associated developing platforms).

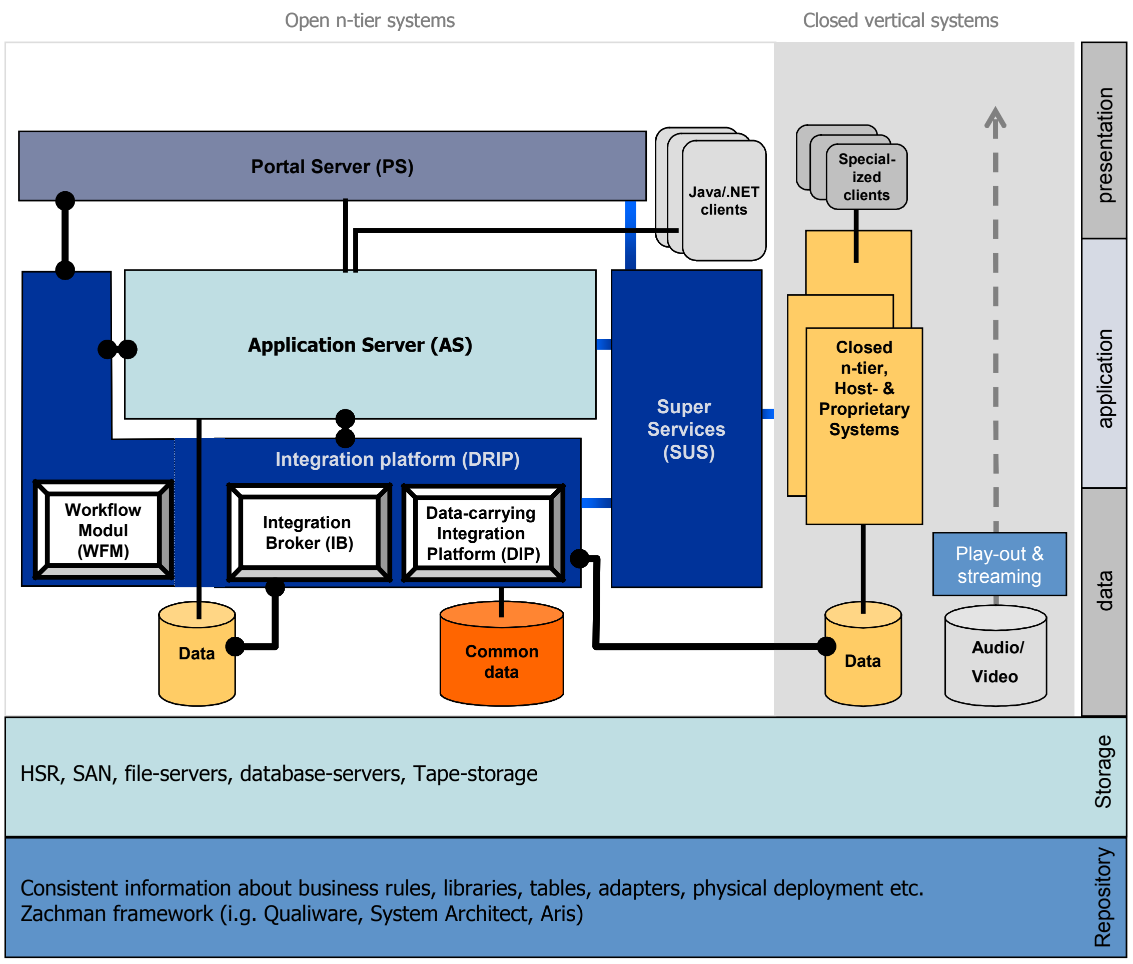

Open n-tier systems
Closed vertical systems
Portal Server (PS)
Java/.NET clients
Special-ized clients
Application Server (AS)
Super Services (SUS)
Closed n-tier, Host- & Proprietary Systems
Integration platform (DRIP)
Workflow Modul (WFM)
Integration Broker (IB)
Data-carrying Integration Platform (DIP)
Play-out & streaming
Data
Common data
Data
Audio/ Video
presentation
application
data
HSR, SAN, file-servers, database-servers, Tape-storage
Storage
Consistent information about business rules, libraries, tables, adapters, physical deployment etc.
Zachman framework (i.g. Qualiware, System Architect, Aris)
Repository

Figure B.4 **The concept level of the integration architecture in our real world**

A combination of three basic methods is used: Message oriented integration using the broker and data-carrying integration platform, Data-carrying integration, and Service Oriented Architecture (SOA).

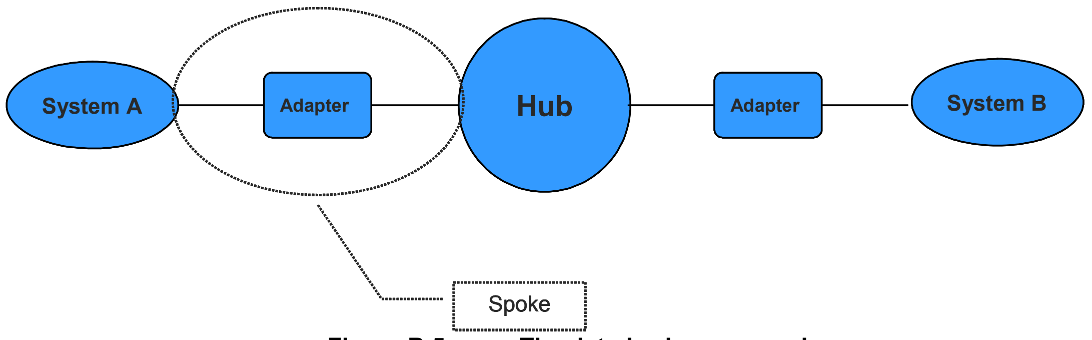


Figure B.5 **The data-broker approach**

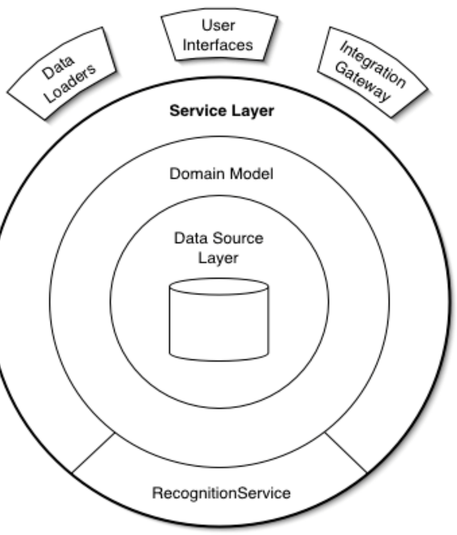


Figure B.6 The SOA approach

The two approaches are working together:

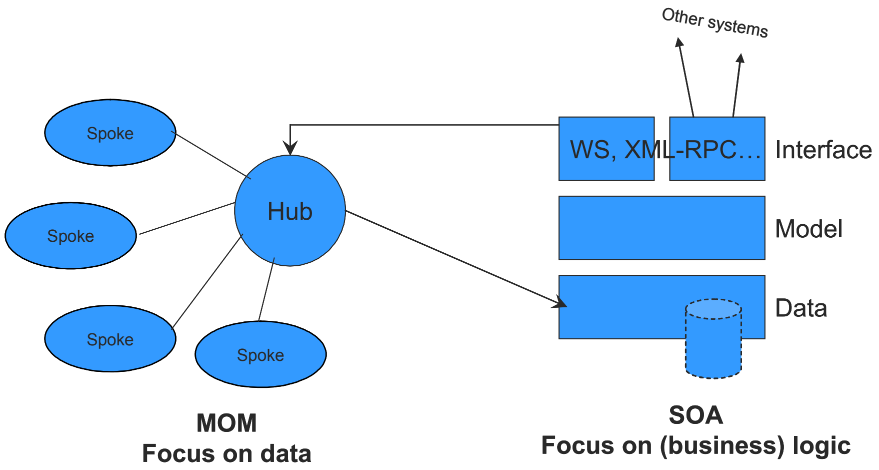


Figure B.7 MOM and SOA combined, using, for example, SOAP

This is the start of implementing the 5 required integration methods:

- **Message based integration** (copying data between systems)
- **Data-carrying integration** (Different systems access to the same data)
- **Portal integration** (Wrapping of user interfaces to the systems)
- **Workflow** (Mechanisms to co-ordinate events in systems)
- **SOA** (Service Oriented Architecture)

The formalisation of the use of portal integration and workflow engines is in its early days; even though there are already defined modules of the integration platform.

B.2 Related issues

The work with a system integration architecture in DR is closely connected to the development and implementation of a DR Metadata model. This was realised already in the above-mentioned strategic work in 1999 as a general need for investigating the handling and control of Metadata in DR. The relation between these two projects can be illustrated as follows:

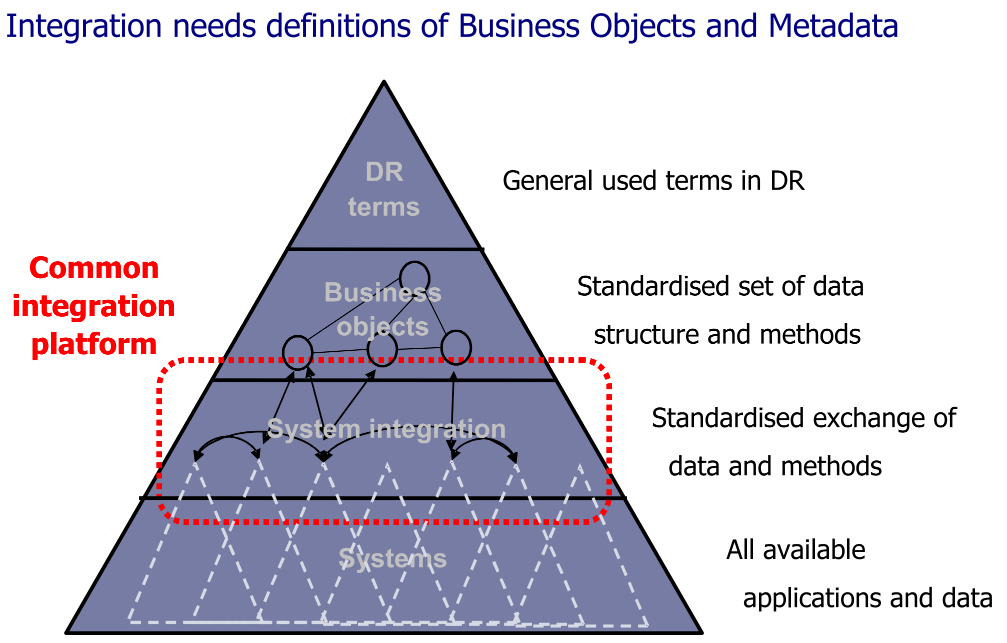

Integration needs definitions of Business Objects and Metadata
DR terms
Business objects
System integration
Systems
Common integration platform
General used terms in DR
Standardised set of data structure and methods
Standardised exchange of data and methods
All available applications and data

Figure B.8 Integrations and definitions of terms and objects

As an example, this shows the present relationships between production and play out:

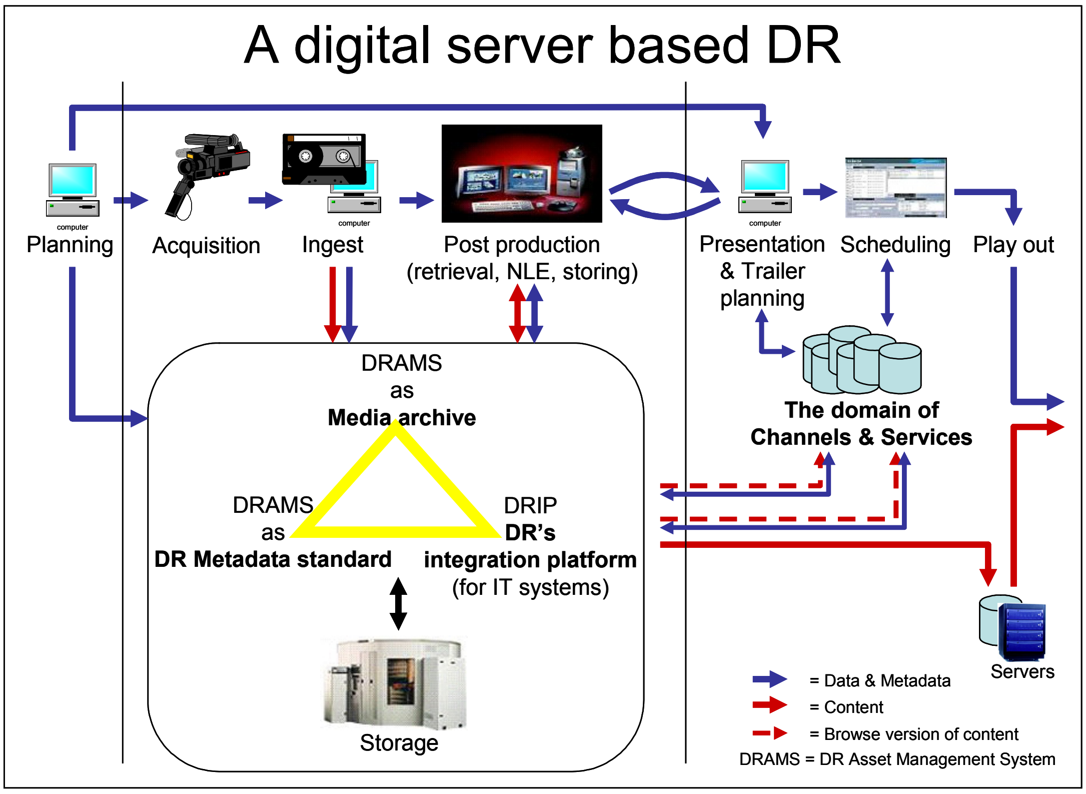

A digital server based DR
Planning
Acquisition
Ingest
Post production (retrieval, NLE, storing)
Presentation & Trailer planning
Scheduling
Play out
DRAMS as Media archive
DRAMS as DR Metadata standard
DRIP DR's integration platform (for IT systems)
Storage
The domain of Channels & Services
Servers
= Data & Metadata
= Content
= Browse version of content
DRAMS = DR Asset Management System

Figure B.8 The main flows from acquisition to play out in relation to the DR asset management

The implementation of our media archive DRAMS is a combination of the actual archive, asset management and storage, the integration platform, and the Metadata model.

During the last couple of years, considerable workflow analysis has been carried out. The need for a quicker and more structured way of securing the description and documentation was needed. Several tools were tested. The result was the decision to acquire a tool focused on system development and quality assurance and the ability to make process and workflow descriptions as well. This tool is also used to host and control versions of the DR Metadata Standard and forms a foundation for building our repository, which is starting to describe the rest of our system and integration environment. The advantage is that both the projects concerned with the business process redesign and the derived change, development and implementation of systems can use different views into the same base of updated information.

Further work has to be done to migrate existing information into this framework.

All in all, this involves huge effort including development of skills and competencies across many areas of our organisation as well as establishing management and control mechanisms related to investments, project management and financials, etc.

Annex C BBC project - detailed description

C.1 Moving the BBC towards IT based Production

BBC R&D has been proposing middleware technical architectures for some time. Initial work to define the ideas has been completed and, at present, many other BBC departments are contributing to rolling out the concepts, mainly as initial trials. We expect the full development of IT server-based production with new workflows and working practices, and no tapes, to be phased in over several years.

In some cases the initial systems are not based on middleware architectures, but, in common with other industries, the use of enterprise scale middleware is a strategic goal and probably a necessity to fulfil the vision.

The BBC makes and distributes TV, audio and new media content, and we are already commissioning material based on content rather than 'TV programme slots'. This requires production teams, with varied skills, to all use common input content, feeds or rushes even if it is re-authored for the many output media that are now available to us. Many laborious tasks like tape copying can be removed if we can hold the media assets on a server in a form that makes them available to the teams producing, in parallel, the TV, web, interactive, WAP, broadband, DVDs and books etc…

This scenario is from where the big savings come, as re-use of material, including rushes that were previously discarded, and the removal of laborious tasks is an obvious advantage both for the producers and also financially.

There is also the potential for an increase in creativity as, given the media assets and appropriate tools, some producers will invent new ways of using the material and new programme ideas that the engineers have probably not even though of.

The engineering goal is thus to build an infrastructure that supports both tapeless operation and makes the media available to the whole production team in an open way that makes it easy for them to process the media to suit their proposed output format. This must be combined with business-orientated Metadata that is common to the whole BBC enterprise.

We still expect manufacturers to produce and innovate their media processing applications and this is encouraged, providing these applications can take as their source, and deliver back to us, material of common content that the BBC is holding.

C.2 Enterprise overview – Two types of middleware identified

The BBC has a large number of production units around the UK. These are distributed round individual departments, with which they are connected and they also require connectivity to independent producers. An important early lesson is that the requirements of each production unit are so diverse that it is unlikely that one simple system, or even technology choice, will suit all requirements.

Our thinking from R&D projects over many years is to divide the problem into several smaller ones by proposing **production workgroups**, each of which serves the production for one programme or strand. These production workgroups can also operate standalone, which gives a reliability advantage, but are also joined up together to form a common infrastructure for all the relevant shared facilities.

There is another practical reason for the production workgroup, which is that the bandwidth required within the workgroup for streaming high quality video is considerably higher than that which can be currently provided by an economic, enterprise-wide network. This does require the network to be connected up and configured for this purpose but as network bandwidths increase this restriction may be removed.

Our requirements for middleware solutions are therefore twofold:

1. Middleware within a **production workgroup**, with components, objects or services representing individual broadcast production functions. Examples could be VTR control, ingest, local data storage for logging and editing, conforming, effects and quality checking.
2. Middleware between **production workgroups,** which provides enterprise level functions such as business Metadata handling and delivery mechanisms to co-producers and playout stations.

We have no requirement that the same middleware technology is used for both these, and indeed individual production workgroups themselves can use different middleware technologies. This is a practical observation that different production workgroups may well be provided by completely different manufacturers.

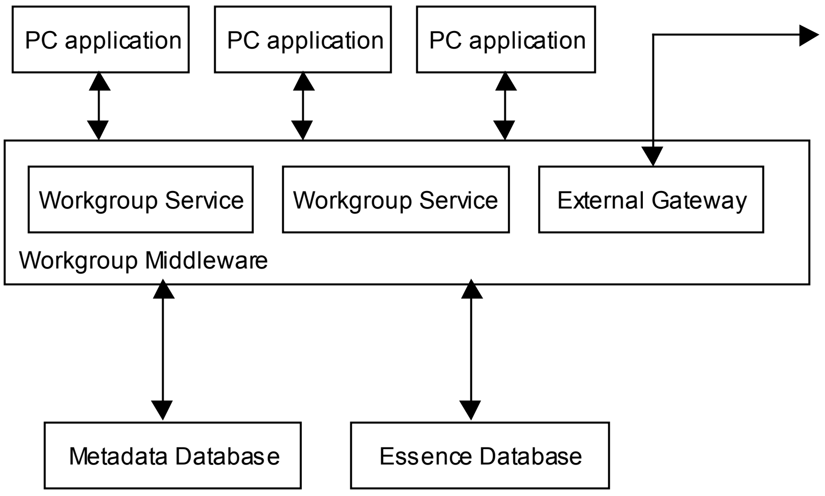


Figure C.1 The middleware for a single production workgroup.

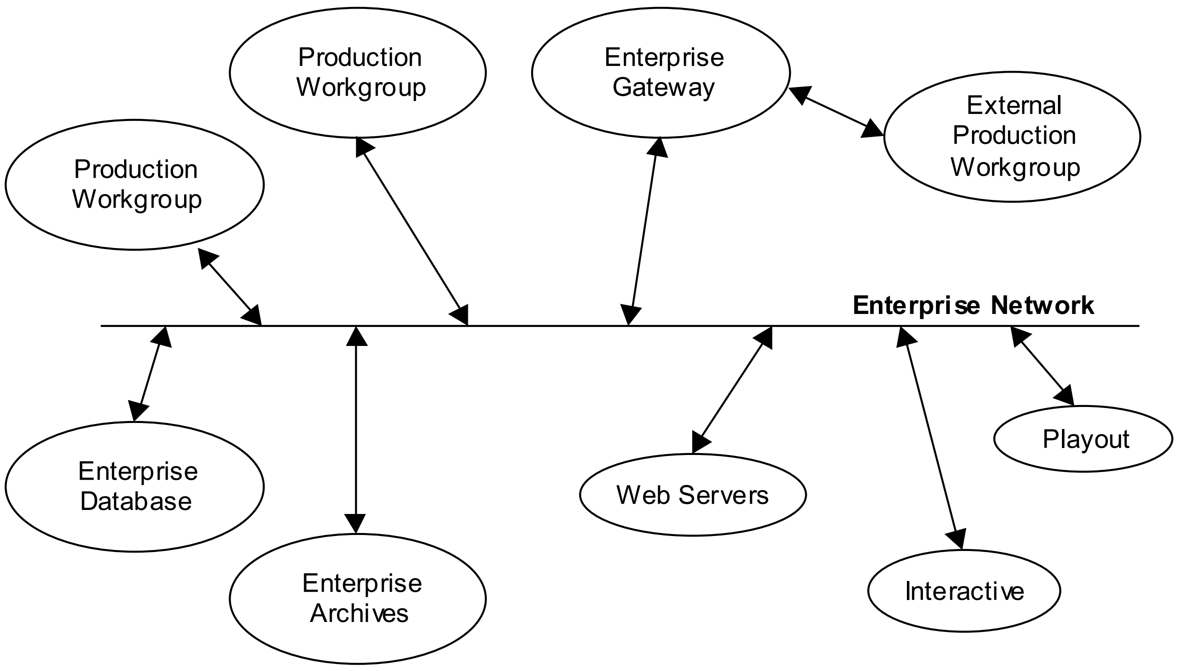


Figure C.1 The middleware for enterprise communication.

C.3 Some BBC projects to develop the architecture

BBC R&D investigated the initial technical concepts in the late 1990s. This has been done by a succession of projects, each with a different focus. In the early 2000s there were projects initiated in the wider BBC to pilot the ideas on production teams. This process continues, and is now part of a BBC-wide long-term strategy to modernise our production techniques. The overall work to modernise BBC production into IT based techniques comes under the One Vision project to which a large number of BBC departments now contribute. The rough sequence of events is documented below:

1. The Atlantic Project (BBC R&D with European partners)

This project investigated the use of IT techniques and their applicability to production on commodity computers. In particular it explored the use of Long GOP MPEG as a suitable format to economically hold enough full quality media and processes, using low cost computers, networks and storage.

2. The Orbit Project (BBC R&D with INESC as partners)

The Orbit Project directly built a 'proof of principle' demonstrator of middleware technologies for a production workgroup. This was a joint project between BBC R&D and INESC Porto.

ORBIT contained the infrastructure of ingest, a separate Metadata and Essence database, and essence location services. The principal of ingesting material once, and then re-using it many times from a common database was established.

3. The Mercury Project (BBC Production Modernisation)

It is no good putting the technology in place if it does not suit the production process. The BBC's Production Modernisation department ran the Mercury project in order to investigate and propose new workflows that would save effort and enable greater creativity on behalf of production staff. In order to do this a technology infrastructure was required and the project commissioned this from BBC Technology. Some main conclusions were that savings can be made and that these are mainly to do with easy access to material and the reuse of material, which requires good logging and database retrieval tools. The benefits of the 'ingest material once into a server, removing later reliance on tape' approach was confirmed here.

4. The Desktop Production Project (BBC R&D)

The Desktop Production is the name given to the project at BBC R&D that covers the standardisation and enterprise infrastructure media asset management projects and this work is ongoing

5. The One Vision Project (BBC-wide with collaboration from Siemens (formerly BBC Technology))

In was realised that the BBC's long term strategy must involve programme production moving to networked computers, with no tape and lots of opportunity for re-use of material. Also the expectation was that many members of a production term would be able to browse the media for research, logging and rough editing purposes.

For a large and diverse organisation like the BBC this is a significant change and it needs to be carefully managed to ensure that the right training is in place and that staff are comfortable with the new ways of working. One Vision brought together a number of BBC departments to promote this vision and, in particular, new ways of working. Although this project has technical strands, its main aim is to look at the 'people' aspects of these changes.

In order to try out these concepts the BBC needed operational technology to gain experience. This was commissioned and supplied, under contract, by BBC Technology (now Siemens Business Services - Media). This development has since been renamed and formed into the Colledia Production product. This is an example of a production workgroup.

6. Manufacturers Workgroups

Some manufacturers have realised that it is a production requirement to work co-operatively in a managed way and have provided proprietary workgroup solutions. Where possible these have been evaluated by BBC production departments for their suitability. While many perform the functions of a workgroup, few have enough open interfaces to allow them to function well in an enterprise environment. The use of MXF and AAF file exchanges is just beginning to happen, but there is a long way to go before such a workgroup can easily access media assets that have already been ingested in another area of the BBC. The hope is that enterprise middleware, which it must be stressed again could offer different service to those within a workgroup, is a solution to this problem.

7. The Core MAM project

In an enterprise consisting of a number of diverse business units it is inevitable that media assets with be ingested and stored in many different places. The idea of a central store for everything is not practical or cost effective. However we may want to make a significant proportion of these assets available throughout the organisation. The Core MAM project is a proposal to record the location of the assets, and maybe make proxy versions of them, and store this information centrally. This is a practical solution to making common material available to everyone, and at the same time not overloading the network. The overhead in each workgroup is to implement an interface to the core MAM and nothing else, which is practical. The alternative is to insist that each workgroup knows about the interfaces and Metadata in every other workgroup and this is potentially unmanageable.

8. Other BBC Asset Management Projects

There are a number of other asset management projects in progress, which hold particular assets like stills, trailers and the archives. Many of these are run to solve individual problems and in the strategic architecture can be considered as individual standalone workgroups or systems. It is hoped that at some stage these can be joined up with the Core MAM.

Annex D Modelling example

This example provides some guidance for analysing business processes in order to define interfaces. These interfaces should be sufficiently generic to enable standards to be based on them. They should be transferable between different vendors as well as between different organisations using products which implement them. They should not include any organisation-specific issues i.e. organisational concepts such as “Batch Number”, Metadata schemes, etc.

D.1 Use Case Analysis

Performing the Use Case Analysis in order to identify the required services within a service-oriented architecture is the first step.

The Use Case Analysis will not lead to a service definition but it will provide an answer to the question “What problem do we want to solve?” i.e. “What to do?” rather than “How to do?”.

D.1.1 Workflow Analysis vs. Use Case Description

There has been much Workflow Analysis undertaken among the members of the EBU during the last few years. Is it possible to use the results of these analyses in order to design the required interfaces for a service-oriented architecture for broadcast systems? Partly yes, but using the results without performing any abstraction would lead to a solution which would fit perfectly, but exclusively, to the organisation which did the analysis, and for the time the analysis was carried out.

Creating the Use Case Description introduces a level of abstraction that makes the results reusable and transferable. The result of a workflow analysis is usually a mix of the description of “What to do” and a description of “How to do”. It is essential to differentiate between those descriptions in order not to get confused in the analysis stage.

D.1.2 Concrete Process

A Concrete Process describes how a task is performed. It gives an answer to the question “What to do?” as well as “How to do?” since it takes into account restrictions and opportunities of a certain implementation technology and the situation in a real organisation.

D.1.3 Generic Process

Since the objective is to carry out a methodology for service architecture design in order to provide a basis for standardisation of interfaces, the concept of the generic process has been introduced. A generic process as derived from the Use Cases that have been carried out by a Workflow Analysis does not include limitations and opportunities of a real organisation. It is limited to answering the “What to do?” questions. This makes the result of the analysis transferable.

D.2 Ingest Example

The P/MDP Group has obtained Use Case descriptions from the BBC that were created during the specification process of the BBC Jupiter and One Vision project. Since the goal was not to produce the specification of an actual system but to identify the required interfaces in order to perform a certain task, an abstraction of the Concrete use cases was required.

The P/MDP group has worked out some principles for performing those abstractions because within the P/MDP workgroup the objective was to identify the interfaces required in one complete Business Process at most.

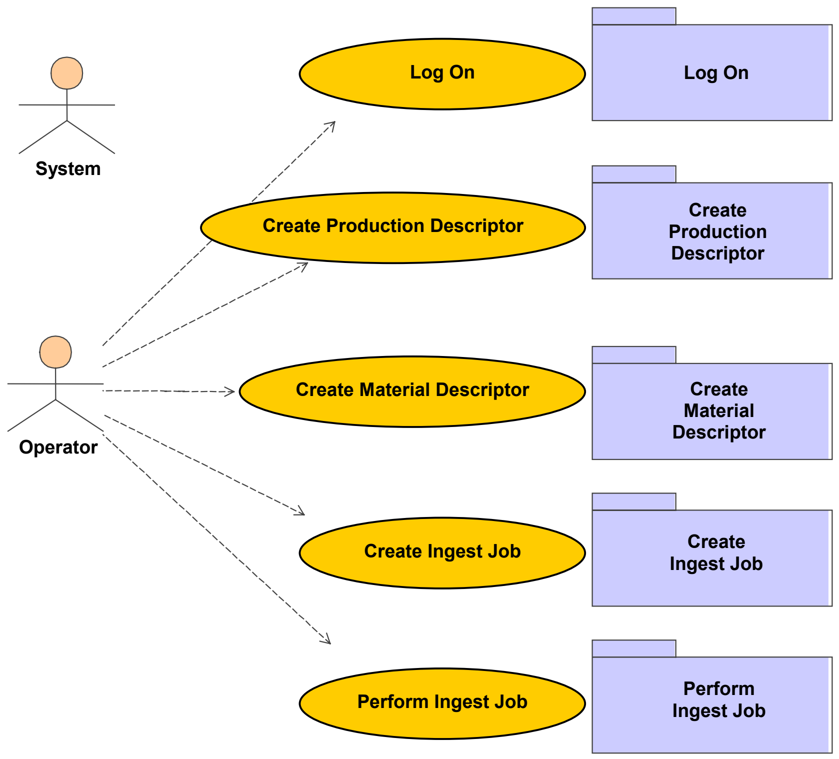

System
Operator
Log On
Create Production Descriptor
Create Material Descriptor
Create Ingest Job
Perform Ingest Job
Log On
Create Production Descriptor
Create Material Descriptor
Create Ingest Job
Perform Ingest Job

Figure D.1 Package View

The package view of the BBC Ingest example shows which packages have been implemented in order to achieve the required functionality. Since the P/MDP Group worked out a Service Oriented Architecture, a system implementing this architecture will likely not make use of the same package structure.

In order to achieve the required level of abstraction the P/MDP group makes use of the concept of generic processes. Generic processes represent the “What” aspect of a system description. This means they describe the essential purpose of a type of processes, e.g. information extraction.

The modelling subgroup of the P/MDP group analysed each Use Case diagram of the Ingest Example and identified the essential purpose of each single sub-process within the Use Case diagrams.

The MDA approach of OMG demands a distinction between domain specific, i.e. broadcast specific in this context, issues of a model and common functionality in general information technology systems.

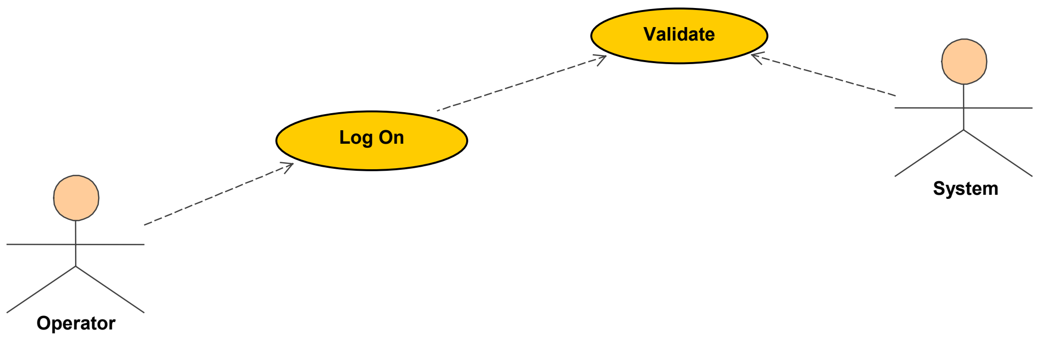


Figure D.2 Log On

The Log On process within the ingest example is not making use of any broadcast specific functionality. For this reason no generic process of the domain model was identified there.

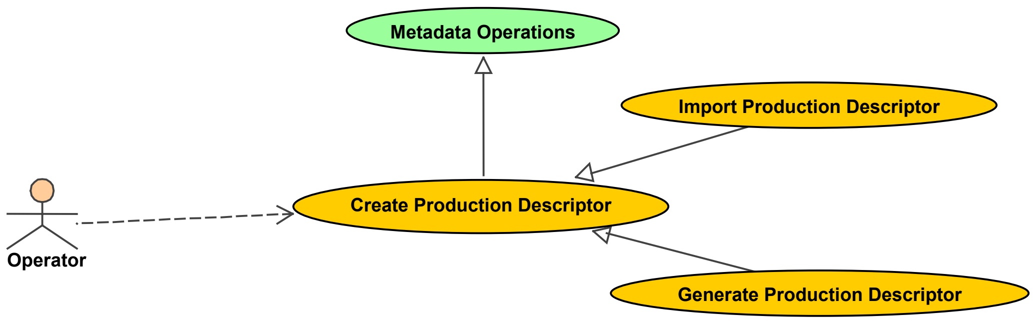

Metadata Operations
Import Production Descriptor
Create Production Descriptor
Operator
Generate Production Descriptor

Figure D.3 Create Production Descriptor

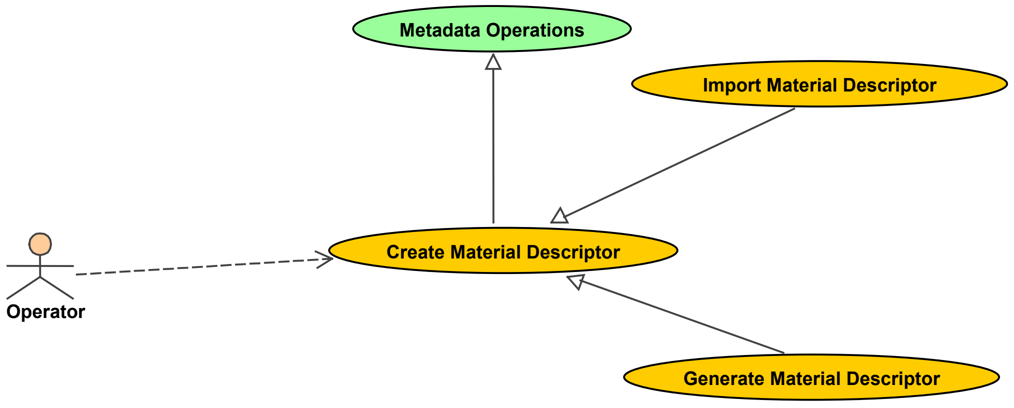

Metadata Operations
Import Material Descriptor
Create Material Descriptor
Operator
Generate Material Descriptor

Figure D.4 Create Material Descriptor

The creation and selection of a “Material Descriptor“ and a “Production Descriptor” is, from a system point of view, a “Metadata Operation”. The service description developed at an EBU level should not be limited to its application within a certain organisation. Since the meaning of the term “Production” may vary among the different Members of the EBU and even within a single Member’s departments it does not appear to be useful to make a distinction between Metadata at a deeper level.

Basically, a selection is a “Metadata Operation”, regardless whether the selection information is stored transiently or persistently, the system needs to store the association between a marker and essence information.

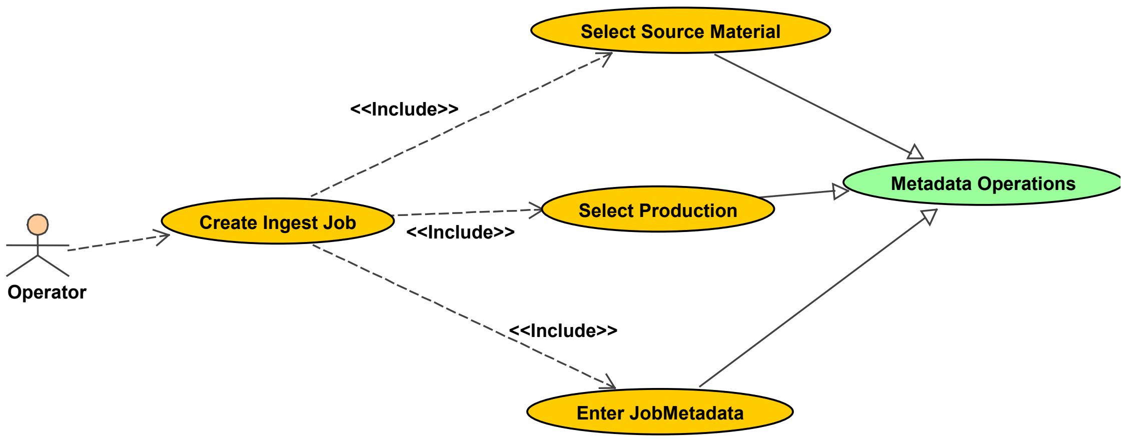


Figure D.5 Create Ingest Job

Selecting material is a “Metadata Operation” as well as selecting a production and entering the job Metadata. Managing workflow as pervasive services as well as maintaining session context is a requirement but should be kept separately in order to achieve a separation of concerns.

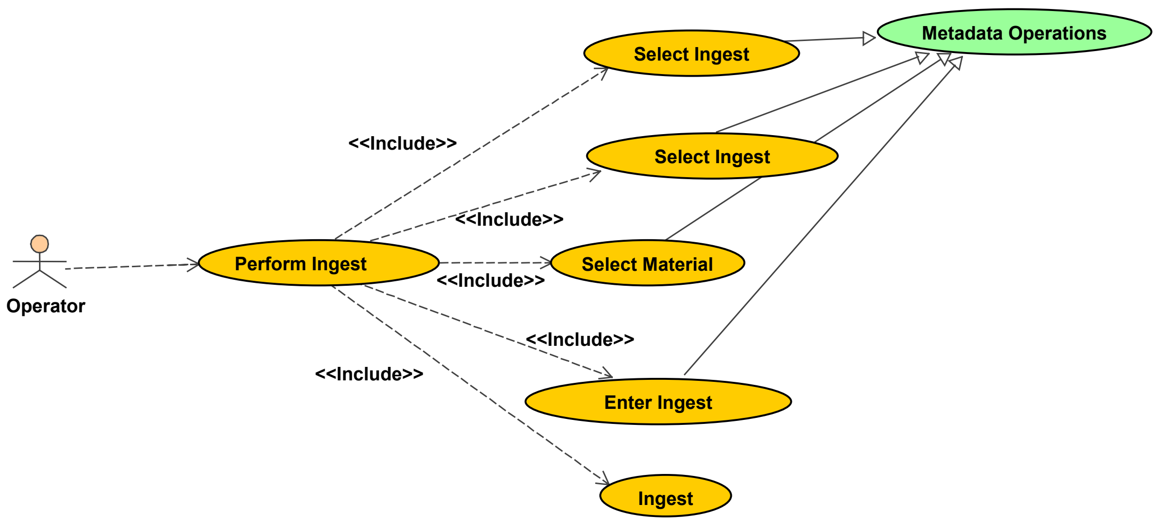


Figure D.6 Perform Ingest Job

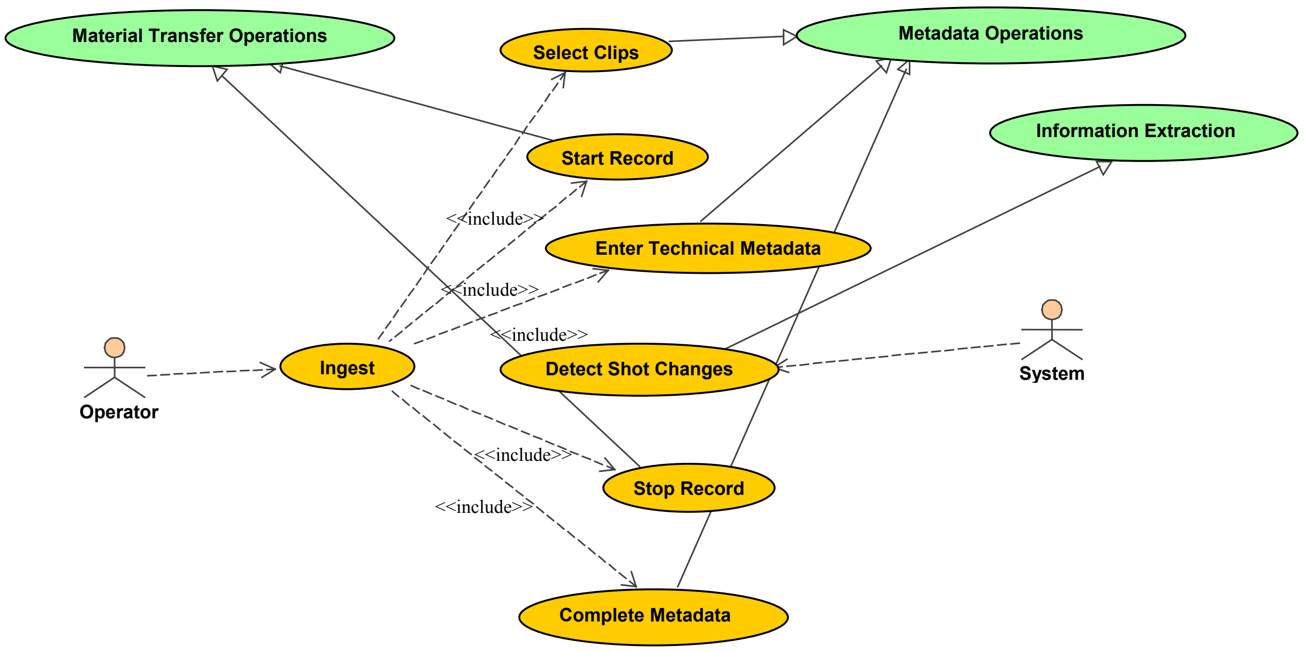


Figure D.7 Ingest

Starting and stopping the record process are operations that do not affect the Metadata; they perform the control on a “Transfer Operation”. There may be Metadata generated when a stop or start command is invoked but these are different processes.

As mentioned before, the Use Case Analysis is the first step. After identifying the Generic Processes, a service architecture has been developed.

D.3 Service Architecture

Three different generic and broadcast specific processes were identified within the ingest example. Now there is a basis to define broadcast specific services of a service-oriented architecture that provides the functionality required to support those processes.

But why is there a concept of generic processes? Why aren’t the generic processes already the service definitions? In general they are not sufficiently concrete in order to do an implementation.

The result of this analysis will not lead to a complete definition of the interfaces of these services because they are derived from one Use Case only. In order to get a more or less complete service definition, more Use Cases (e.g. Play out, Editing, Archiving) need to be investigated and the results have to be verified. The resulting service architecture for the Use Case “Ingest” is shown in Figure D.8.

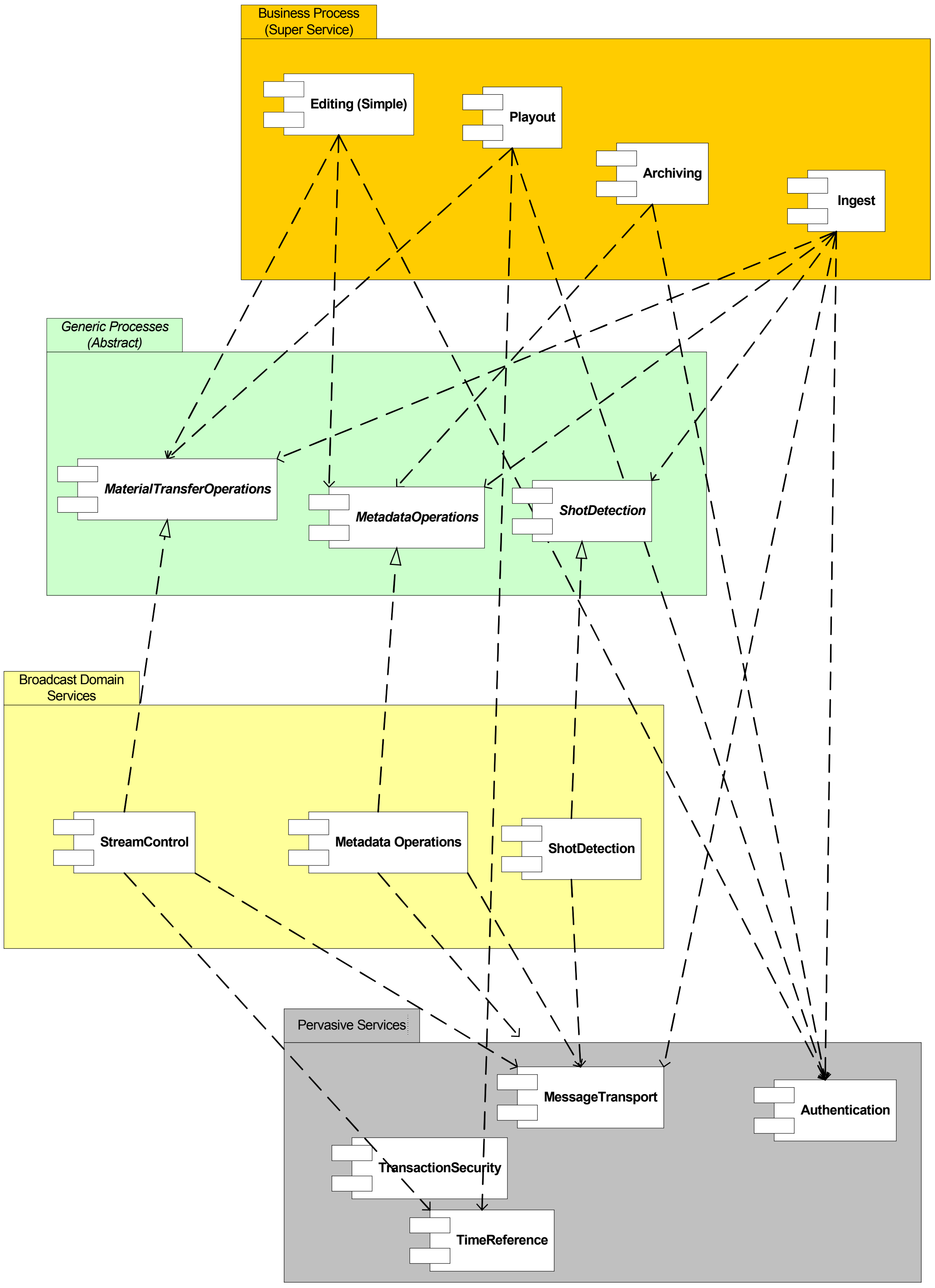


Figure D.8 Service Architecture for Broadcast (analytic concept)

Since the design of the domain model for broadcast shall provide a basis for the standardisation of interfaces it appears to be useful to make use of already existing standards where possible.

The aim of these efforts is to define interfaces on a semantic level rather than on a technical level.

D.3.1 Stream Control

This service may be exposed by any device that is able to provide material streams with a time reference. Any information may be serialised and provided as a stream but this service provides interfaces that allow the control on the stream. For this reason a time reference for the control is required and the material needs to have an intrinsic time reference e.g. frames, samples/s etc.

Play(UID)

Record(UID)

Stop(UID)

D.3.2 Metadata Operations

The Metadata operations service needs to be agnostic to a Metadata model. This is because if the service is limited to one Metadata model the resulting architecture that includes this service definition will not be transferable among different organisations. For this reason the interface definition of the Metadata Operations Service makes use of the UID and SMPTE Universal Labels. This requires a schema to exist in order to define the semantics of any label used.

Create(UID, UL)

LoadSchema(*Scheme*)

Update(UID, UL, *Value*)

Insert(UID, UL, *Value*)

Select(UID, UL)

D.3.3 Shot Detection

Shot detection is a type of information extraction. Simply using the term means that information extraction is not sufficient to define a concrete implementation of such a service. Not using the generic process information extraction would limit the results of the analysis to applications where the information extraction is shot detection and it would not be applicable if the desired information extraction is a transcription.

DetectShots(UMID)

D.4 Conclusion

The modelling subgroup of P/MDP has identified just three generic and broadcast specific processes within the studied example.

Of course these generic processes do not represent the complete functionality of an ingest application. A complete ingest application will make use of services which implement the interfaces derived from these generic processes.

Annex E The Zachmann framework

E.1 Why you need an information repository

When dealing with the management and operation of the system environment, and certainly when dealing with the system integration architecture and its components, different people and work functions have different requirements of the information available to them.

Describing the same level of the system from a single point of view may be done using different tools. A number of structured approaches have been developed, such as the IDEF model for describing workflows and activities, and UML for describing activities, objects and data involved in using a system.

As the landscape changes from mostly isolated systems into a wide and very complex integrated, modular and object oriented system architecture, it may be difficult to maintain an overview of your environment. This is particularly true when different kinds of descriptions are kept in different formats, in different places and in different versions (updates).

Some kind of repository for all the information about your systems is necessary. This should hold information such as data-models, adapters to data-brokers, and a catalogue of web-services used, etc.

E.2 About the Zachmann framework

The Zachmann framework (figure E.1) tries to provide a structured high-level matrix for addressing the many combinations of views, system levels and description tools.

It can be used in several ways:

- As a user interface for a repository (information database) containing all the information about processes, systems, data models, Metadata schemes, deployment, etc. Clicking on a frame will open up a suggested primary set of models and visualisations, which are seen as most appropriate for that combination of tasks and objectives.
- As a checklist, for use during project evaluation, decision making, project management and change management whilst in operation.
- As a way of structuring the responsibilities and roles around systems: process owner, system owner, designer, system architect, programmer, service, helpdesk functions, etc.

More information on the Zachmann framework can be found in [6] (below we provide an outline).

For details on the IDEF framework, see [7]. Information about the UML set of descriptions can be found in [8].

There are two critical, distinctly different challenges facing an organisation (called enterprise in the following) that is going through an integration process. These challenges affect its ability to operate effectively and to dynamically respond to changes.

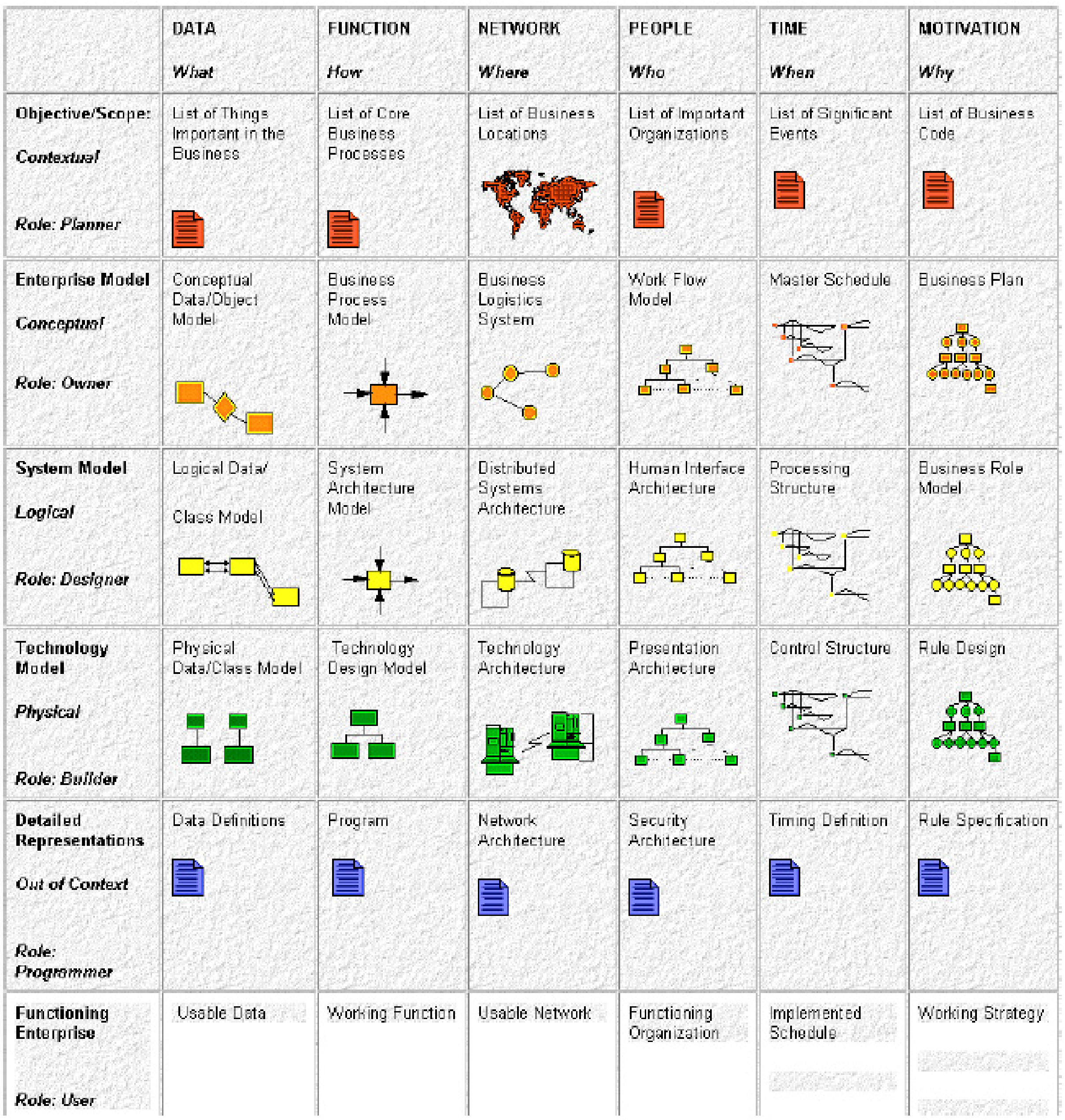
The enterprise should:

1. Begin making descriptive representations (models) of the organisation explicit.
2. Formalise and enhance the enterprise architecture process.

Ad 1: this includes populating various cells of the Zachmann framework with instances of models. For example: defining specific models of the enterprise, with the intent of producing system implementations that more accurately reflect the intention of the enterprise. These models can then also be used as a base-line for managing enterprise changes.

Ad 2: initially this would consist, for example, of defining the generic components (contents) of each of the cells of the framework. In other words: define what is important to capture in each cell when building instances of models for that enterprise. This allows you to formalise the management system (plans and controls) for building the enterprise models, for evaluating and deploying development methodologies and tools, for selecting and/or defining databases (repositories) for storing the enterprise models, and for defining the roles, responsibilities and skills required within the architecture process, etc.

The above two challenges go hand-in-hand. The enterprise must produce the models in order to deliver systems implementations in the short term, and at the same time, for the long term, initiate the architecture process in order to ensure on-going coherence of system implementations and to build an enterprise environment capable of accommodating high rates of change.



	DATA *What*	FUNCTION *How*	NETWORK *Where*	PEOPLE *Who*	TIME *When*	MOTIVATION *Why*
Objective/Scope: *Contextual* *Role: Planner*	List of Things Important in the Business	List of Core Business Processes	List of Business Locations	List of Important Organizations	List of Significant Events	List of Business Code
Enterprise Model *Conceptual* *Role: Owner*	Conceptual Data/Object Model	Business Process Model	Business Logistics System	Work Flow Model	Master Schedule	Business Plan
System Model *Logical* *Role: Designer*	Logical Data/ Class Model	System Architecture Model	Distributed Systems Architecture	Human Interface Architecture	Processing Structure	Business Role Model
Technology Model *Physical* *Role: Builder*	Physical Data/Class Model	Technology Design Model	Technology Architecture	Presentation Architecture	Control Structure	Rule Design
Detailed Representations *Out of Context* *Role: Programmer*	Data Definitions	Program	Network Architecture	Security Architecture	Timing Definition	Rule Specification
Functioning Enterprise *Role: User*	Usable Data	Working Function	Usable Network	Functioning Organization	Implemented Schedule	Working Strategy

Figure E.1 Overview matrix used in the Zachmann Framework *(source: www.zifa.com)*

As an example, Pinnacle Business Group (and its predecessor companies) invested over 15 years in developing a robust methodology including tool support for doing the architecture planning, business process engineering and application development.

Annex F Technologies to come

Broadcasters and manufacturers are moving towards layered and more open and modulated systems. Further down the road some important technologies are waiting, which should be kept under surveillance.

F.1 Grid technology

F.1.1 Introduction

Grid computing is a hardware and software infrastructure that clusters and integrates high-end computers, networks, databases and scientific instruments from multiple sources to form a virtual supercomputer on which users can work collaboratively.

If you can think of the Internet as a network of communication, then grid computing is a network of computation: tools and protocols for co-ordinated resource sharing and problem solving among pooled assets. These pooled assets are known as virtual organisations. They can be distributed across the globe; they're heterogeneous (some PCs, some servers, maybe mainframes and supercomputers); somewhat autonomous (a grid can potentially access resources in different organisations); and temporary [1].

A grid structure requires two key elements before services can be provided. These are high bandwidth network interconnections and grid middleware to facilitate grid applications and services.

Once these elements are in place, grid middleware provides functionality to support grid-based applications and environments. Grid middleware has the following characteristics:

- Geographically distributed data sets
- Geographically distributed resources
- Facilities to move data from point of storage to point of use
- Facilities to manage distributed data sets
- Definition of use of distributed technical assets
- Dynamic resource repositories and resource query services
- Security, authentication and accounting

F.1.2 A broadcasting grid

A grid is a service-based infrastructure that facilitates the management and resource sharing of a networked organisation, and the interaction between that organisation and other networked organisations. It is convenient to structure the organisation as a collection of domains each of which is a coherent organisational unit. This structure is intended to facilitate the identification of domain and organisational services and to focus on security issues within the organisation. An outline grid domain architecture is defined in figure F.1.

The proposed grid architecture (initially) identifies infrastructure domains. Work is currently under way within the UK to develop grid based broadcast services and the BBC is involved in a pilot project to assist this development.

Nation Broadcaster Domain

Provides the storage, control, and technical resources that are required to operate a nation wide storage structure.

Network Control Domain

Provides the storage, control, and technical resources required to manage a network schedule consisting of subordinate broadcast locations.

Production Domain

Provides the storage, control, and technical resources required to manage the production of television and radio programmes.

A broadcasting grid is intended to enable the management of the inter-relationships between these domains and to enable the sharing of data and technical resources between the domains. In grid terms each domain is defined by the services that are provided to other grid domains.

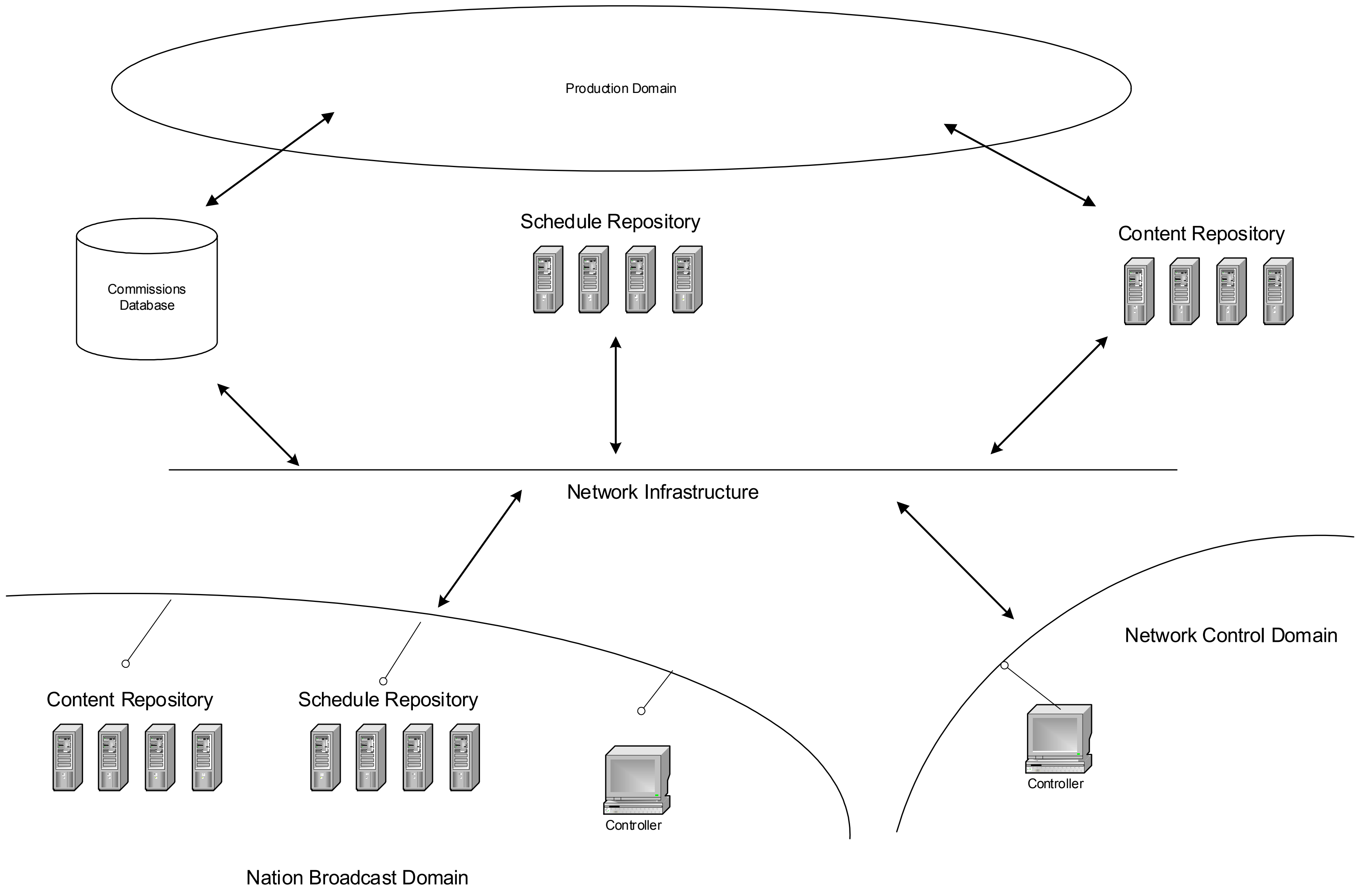


Figure F.1 A broadcasting grid Domain Architecture

Each domain may define its own grid infrastructure to enable the management of resources within that domain. Thus, the production domain may define grid-based production grid services and workflows to manage the development of programme and radio content. A national structure may define a local broadcasting grid to facilitate local production and content creation, schedule creation and broadcasting.

These domain grid services and sub-grid domain services may form their own virtual organisations with compatible domains, sub-domains or third party grid infrastructures. The sub-domains and their constituent services are the focus for domain-based security and management. Thus, the particular production grid services may inter-operate with services within another production domain, with production domain services visible with greater transparency than those in other domains. Or, a production domain may form a virtual organisation with an external production company to enable content and resource sharing.

Similar virtual organisations may be formed to control and manage broadcast schedules or content archiving. In the grid pilot the emphasis is on identifying the services that domains and sub-domains required, the inter-related virtual organisations that are required or convenient for domains, and the security and service infrastructure required to achieve content sharing and resource use.

F.1.3 Domains and sub-domains

A grid domain is structured as a collection of sub-domains with:

1. public domain-based services to provide services to other grid domains;
2. private domain-based services that permit resource use and management of the domain grid services; and
3. collections of sub-domains structured in the same manner as a grid domain.

Grid domains thus form a hierarchy with increasing business focus and speciality. For example, a grid for a broadcast nation domain might be structured as illustrated in figure F.2.

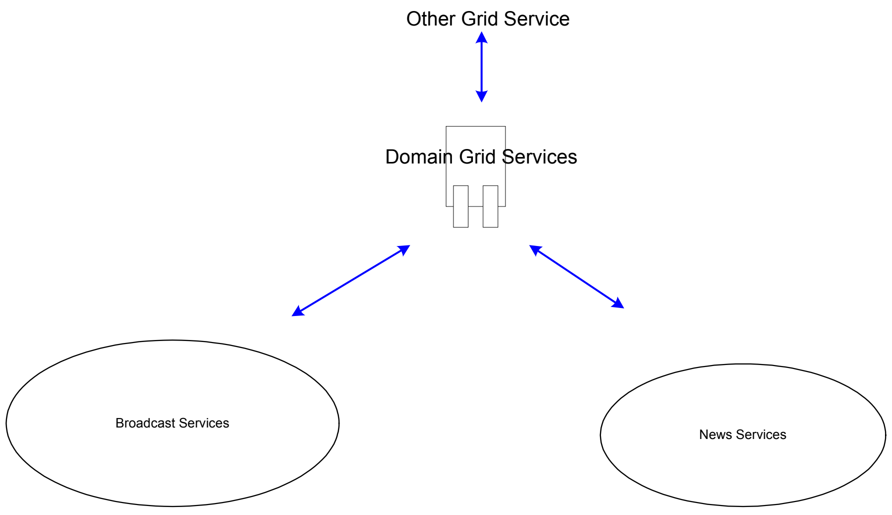


Figure F.2 An Example Broadcast Nation Domain

F.1.4 What makes up a domain?

A domain is defined by its service interface to other grid domains, its internal service infrastructure and its internal collection of workflows.

Grid workflows perform (at least) two distinct roles-:

- to capture business process operations (c.f. Lock workflow below); and
- to provide a bridge from grid-based standards and operations to broadcasting or proprietary standards and operations.

A Content Domain, for example, may be defined by the following services:

- *Add* : that submits a new item to the content repository;
- *Withdraw* : that removes content from the repository;
- *Lock* : that prevents broadcasting or additional copying of stored content;

- *Copy :* that requests a copy to be stored in a remote repository; and
- *History :* that requests data on the history of stored content;

and by workflows that

- enable automatic duplication of stored content to facilitate resilience and access load balancing (A *Resilience* work flow);
- automatically copy content to remote repositories that have requested the content when that content becomes available (An *Addition* Workflow); and
- prevent broadcasting of content locked by a user (A *Lock* Workflow).

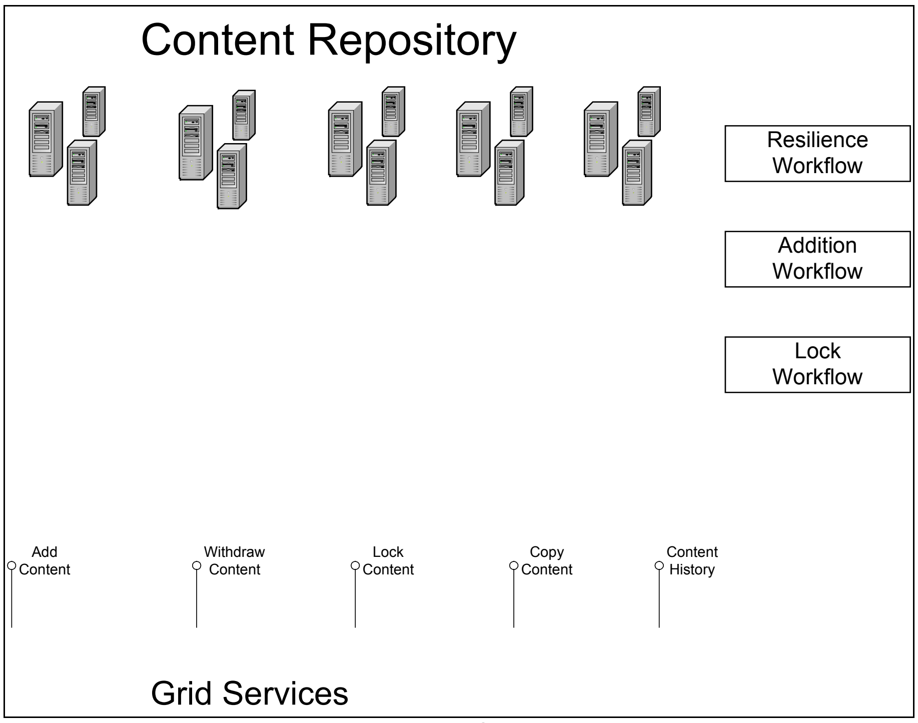


Figure F.3 A Content Domain

F.1.5 Boundaries between domains

The boundary for a grid domain is defined by the services provided by that domain to other grid domains. (To other domains, a domain is its collection of services.) A grid consisting of a collection of domains is defined by its collections of services provided by its constituent domains.

In addition, a grid will require shared data that is accessible to multiple domains. Each coherent shared data set defines an additional domain and thus its collection of grid services and internal workflows.

The grid architecture in figure F.1 defines three additional data domains:

Commissions

A record of television and radio programme Metadata.

The grid role of the *Commissions domain* is to provide additional Metadata on programmes to be broadcast.

Content

A repository of radio and television programme content for broadcast or production purposes.

The grid role of *Content Domain* is to manage access to and availability of broadcast content.

Schedules

A repository of radio and television schedule data for broadcast or production purposes.

The grid role of *Schedule Domain* is to manage access to and availability of schedule information.

Each sub-domain or sub-grid will define its own shared data elements and these elements may form their own virtual organisations with related grid domains or sub-domains. Thus, a grid schedule domain may form a virtual organisation with larger schedule domains.

F.1.6 Boundary workflows

The broadcasting grid pilot currently being run within the BBC is concentrating on supporting the broadcast operations of a BBC Nation and measuring the practicality of grid-based broadcast infrastructures by developing Quality of Service data for a broadcasting grid. There are, however, overlaps between domains that affect the way in which broadcast grid services can operate. A project aim is to identify boundary workflows and defined their relationships.

For example, consider as a boundary workflow: *the Availability of Content*

A production domain actor releases content that has been completed. A boundary workflow updates the appropriate Metadata that is part of the Commissions domain and either copies or unlocks the content stored in a content domain.

Questions:

How is content yet to be produced referred to in schedules?

Is the availability to be triggered only by the update of Metadata?

Is production and completed content stored in the same repository?

F.1.7 A service network architecture

A grid infrastructure does not prescribe a network topology. However, a grid infrastructure does assume that a network architecture has been defined that enables fast, reliable inter-operation of grid-based services. (This requirement means that high speed geographically dispersed service interaction is possible.)

The requirement for fast, reliable service interactions is often interpreted as requiring high-bandwidth IP-based network connectivity between grid services. For most application areas the provision of high bandwidth IP-based network connectivity would be the grid implementation solution and it will be a foundation of the BBC's broadcast grid pilot architecture. There is, however, no requirement in a grid-service infrastructure for there to be only one network that carries all grid service communications. Indeed, access to 3rd party networks may provide two or more routes between domains in a broadcasting grid.

It is convenient in analysing a potential broadcast grid to consider the potential network traffic in the broadcast grid. Three broad traffic types can be identified.

(a) General Purpose Traffic

Typically, this is stored content being transmitted between broadcast sites for broadcast or production and grid service traffic to enable remote service provision.

(b) Short life-cycle Traffic

Typically, this is stored news and trails content for short-term broadcast use and content sharing between broadcast and production sites.

(c) Time Critical Traffic

Typically, this is live content that is being multicast to BBC Nations for transmission.

Type (a) does not have any significant QoS requirements and is suitable for a high-bandwidth IP-based network solution. Type (b) is a storage network that enables stored content sharing between distinct grid domains and is suitable for a generic SAN solution. Type (c) has high QoS requirements and a network solution that makes these QoS requirements achievable.

The proposed network architecture, figure 4, partitions a grid network into three distinct network spaces to accommodate these distinct network traffic types. This partitioning may be physical and employ three network solutions or be managed bandwidth within a single network solution.

A grid service infrastructure provides a means of managing these distinct types of traffic and their inter-relationships. For example, short life cycle content may be migrated from content sharing area to long-term content storage using automated content grid workflows. The control of the live content network will be by grid service interactions on the general-purpose IP-based network.

F.1.8 Links

For more on Grid middleware, see for example:

- Globus GT2 [8]
- OGSA/OGSI overview [9] and anatomy [10]
- The application examples in [11] and [12]
- Other Grid Middleware, such as Legion [13] and Unicore [14]
- Grid-enabled CORBA [15]

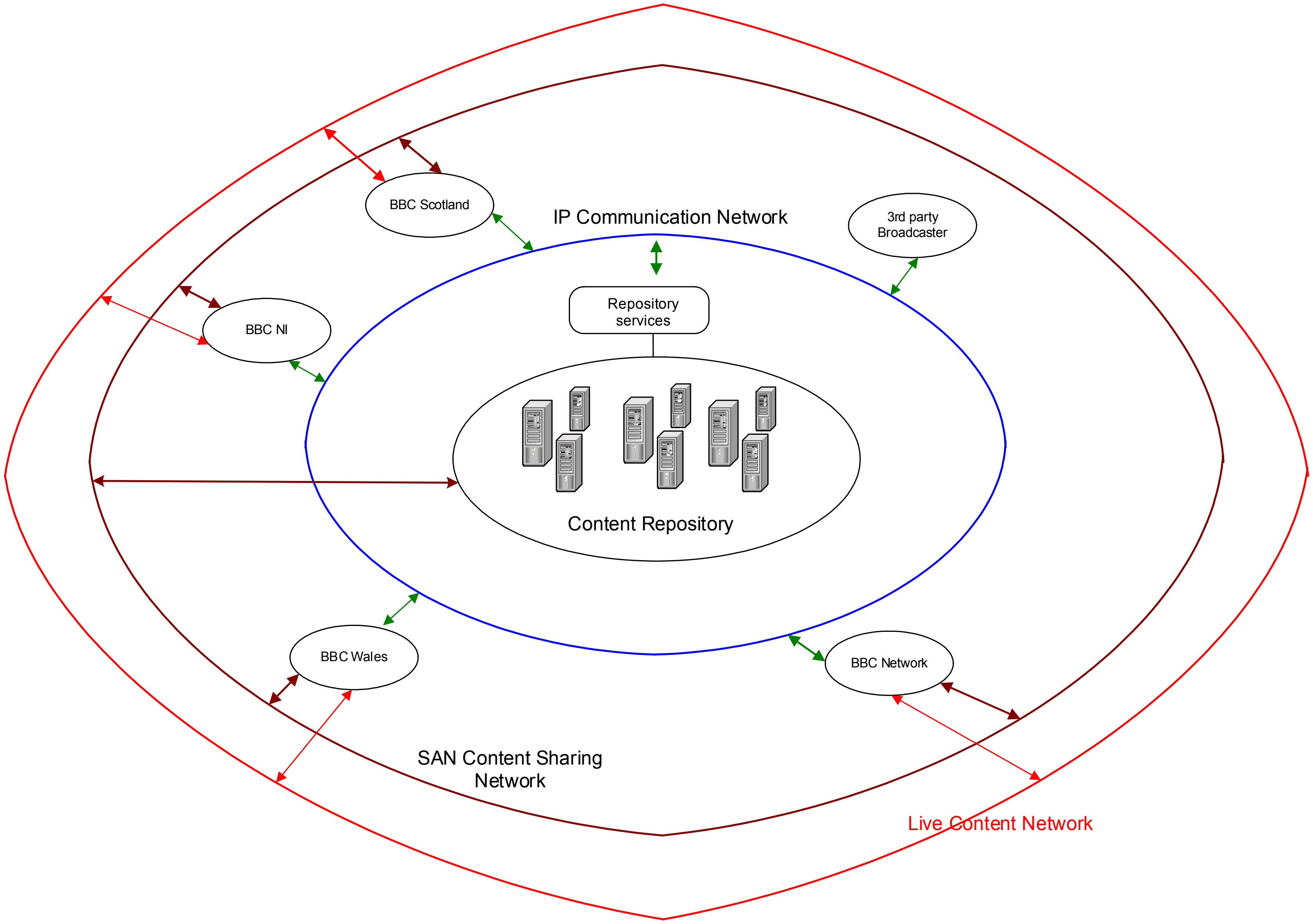


Figure F.4 An example network topology

F.2 Meta-Metadata

F.2.1 Introduction

Be gentle please.

One of the core aspects when dealing with a system integration problem is usually the design and implementation of a data integration architecture.

As explained in chapter 2, from the point of view of a system integrator, any system may be modelled as a set of organised components, each with a well-defined interface. The definition includes the interface's identification, I/O parameters and functions, behaviour and semantics.

Typical tasks that middleware components have to perform are:

- Adaptation of the meaning and structure of the I/O parameters passed to and received from the interface functions of the components.
- Adaptation and extension of the behaviours of the individual integrated components to achieve the new goals of the integrated system.

F.2.2 Where do we need meta-Metadata?

Wasn't ordinary Metadata already enough of a problem?

A problem in knowledge exchange

Practical experience shows that systems designed and 'grown' in different contexts (with different aims and conceived by different design teams) implement incompatible data models. That is, they use different data structures to represent the same domain.

For example: two different Content Management Systems may represent their media assets using a different terminology, a different set of classes, and a different set of relationships among the classes, yet being both are aimed at the management of the same kind of real objects.

If we wanted to set up an information exchange mechanism between the two systems, we would have either to elect one of the two models as the intermediate model for the exchange, or to work out a third model as a synthesis of both sides.

The design of an intermediate model for information exchange is, in general, a complex task. Especially when the number of systems potentially being involved in the exchange grows. In fact, if we consider a complex system consisting of N subsystems and want to develop a distinct interface for each distinct pair of communicating subsystems, we would need N(N-1)/2 distinct interfaces.

Using a meta-model

OK, but now propose a solution.

How could we simplify and manage the problem outlined above? One way to avoid these kinds of complexities is to establish a "hub-and-spoke" environment. In our context this would mean creating a central neutral metamodel, able to represent and carry the knowledge of all the systems without any loss of information.

The term "metamodel" naturally evokes the concept of being "beyond" the individual models. The key aspect is that such a model should make explicit what usually is implicit in the individual realisations, by means of defining and exchanging instances of high-level categorised information, thereby acting as an abstract super-class hierarchy for all the possible classes of the domain.

An example of a possible metamodel categorisation could be as follows:

- Types
    - Concepts
        - Works
            - Literary Works
            - Art Works
            - TV Programmes
    - Events

        - Real life Events
        - Media Events
        - Broadcast Events
    - Essences
        - Artefacts
        - Persons
        - Tapes
        - Files
    - Properties
        - Descriptors
        - Identifiers
        - Classifiers
    - Groups
        - Organisations
- Relations
    - Description
    - Identification
    - Classification
    - Derivation

The responsibility of each system taking part in a knowledge exchange would be to provide the mappings between its data structures and the meta-structures present in the agreed meta-model.

The typical result would be the definition of a standard EDI interface providing the specifications of the document templates suitable for the related information exchange (e.g. an XML Schema or an RDF vocabulary), plus a set of export/import methods from the individual data repositories for use with the common format. The idea is illustrated in the following figure.

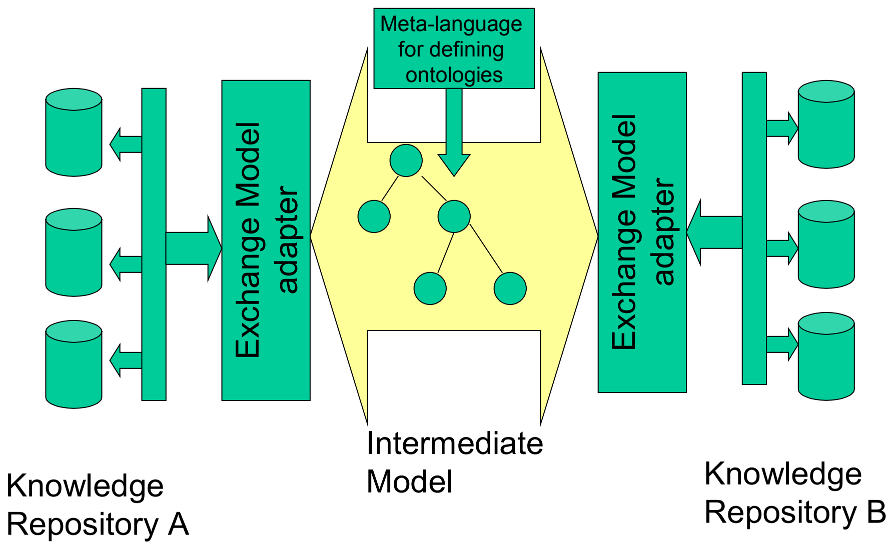


Figure F.5 A meta-model based exchange framework

Workflow management

Sooner or later, we always end up talking about workflows!!

Another issue related to information representation heterogeneity is the lowered ability to set up flexible workflow management systems when a diversity of data schemes is used for the information flow in the value chain.

Changing the on-going behaviour of up-and-running systems may prove to be unaffordable, because the workflow business logic is very often locked into the processing logic of the applications taking part in the chain (e.g. any application knows exactly on which data item it has to operate. Any change in the data would require a change of the logic of some or all applications).

To solve this problem, meta-Metadata may be used. The meta-Metadata would comprise terms of annotation that describe the crucial characteristics of Metadata with respect to whom, when and how they should be used, managed or instigated, and also in terms which identify the nature of the Metadata items.

Consider the following categories:

- The Metadata item importance, e.g. essential, optional, recommended.
- The moment in the process where the Metadata should be instigated, checked, etc. For example: planning, editing, archiving.
- The workflow actor in charge of using, instigating or checking the Metadata item, e.g. the commissioner, the producer, the editor.
- The type or nature of the Metadata item, e.g. a descriptor, an identifier, or a generic attribute of an event.

Figure F.6 illustrates the relationships between the definition of these parameters and the generic actors in the production workflow.

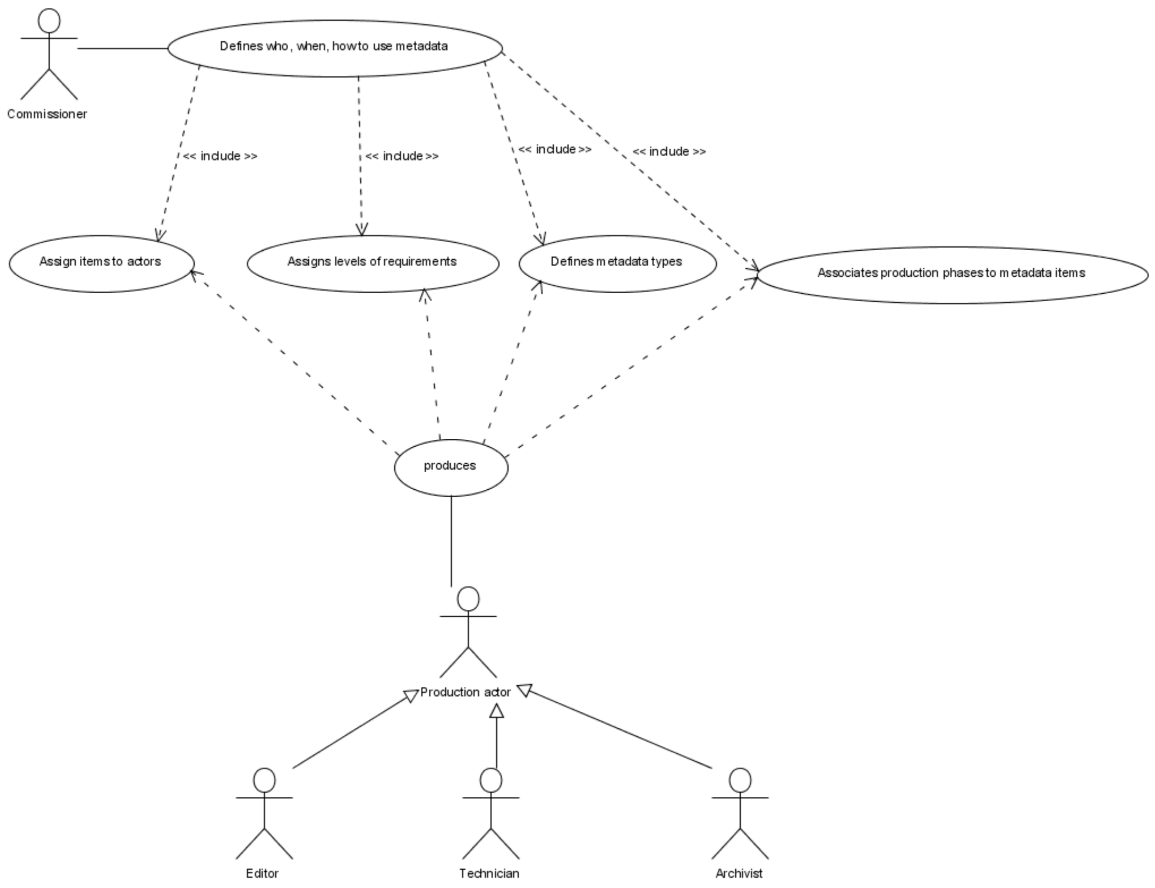


Figure F.6 Metadata characterisation and workflow

F.2.3 Why do we need something new?

What's wrong with using current Metadata standards?

The most important reasons for needing 'something new' are:

Growth of information sharing

In the near future we will see a continuous growth in the demand for sharing information among increasingly heterogeneous contexts (digital terrestrial and satellite broadcast, paper, Internet, new personal communication media).

Generally, existing Metadata standards are too restricted to model and manage a single well-defined domain, and they don’t work well with cross-domain problems.

Automatic tools using meta-models

The existing and on-going developments of automatic searching and reasoning tools are based on meta-models, or at least on their natural translation in that community, i.e. ontology.

Effective broadcast media production models can reasonably exist if based on automatic processes as much as possible.

Flexible workflows

Flexible workflow models are fundamental for the exploitation of traditional broadcast assets into the new media world, where the evolution of the communication languages, presentation structures and delivering media are incomparably more dynamic and changeable than in the usual TV and Radio broadcast scenario. Ability to adapt workflows based on market response is the basis for the effectiveness of these exploitations and for an actual return on investment.

F.2.4 What currently exists?

Is someone doing all the work already?

Widespread efforts have been taking place in this interesting direction for several years and it is expected that these efforts will increase.

The most important initiative in this field is without doubt that currently being developed by the W3C consortium, called the “Semantic Web” [16]. The core technology in this research area is represented by RDF (Resource Description Framework), a W3C standard initially intended for the description of the resources distributed over the Internet, and now becoming the kernel language of any advanced system for knowledge representation. Some of the most important languages developed on top of RDF are DAML (Darpa Agent Markup Language) and OWL.

The OWL language (Web Ontology Language [17]) is a RDF-based language for the definition and management of ontology. In the semantic web context “ontology” is a structured dictionary of terms indicating classes of individuals and relationships that can be established among these individuals. Ontologies can eventually be represented and exchanged in the form of XML documents and can be processed by reasoning agents able to extract additional information from these documents.

Recently the broadcast community has begun to invest resources in this area. In particular, the RAI has developed a proprietary framework for ontology-based exchanges of information named MADL (Media Asset Description Language) with which to implement all of the internal business information exchanges.

Also, BBC R&D has developed an original view on the topic, called the Metadata GK (Genre/Kritikos) Scale. This initiative has defined three sets of meta-Metadata for the characterisation of genre-critical Metadata items with which to achieve flexible management of the production workflow. An example of one of the tables is shown below. The other two tables provide information on the 'who' and 'when' aspects of the Metadata.

The purpose of this additional information is to provide links with the business layer to enable flexible process management when common systems have to deal with different genres of programme making. The business can develop required GK data sets for different Genres and programme producers that can be applied at the programme commissioning stage.

This allows the business to provide a range of managed processes that can be modified or extended by agreement, without the need to redesign process applications and user interfaces as programme making requirements evolve.

The GK Scale could also provide a method to allow businesses to customise common user interface and system management products to suit their particular needs without expensive design modifications providing products are designed to respond to GK data.

Essential	Must be present – cannot proceed without this
Highly Recommended	There should be a good reason why this was not entered
Recommended	Ideally present, but if missing can still continue
Optional	Enter this if known
Not Recommended	This should not be entered, but if already imported no problem
Cannot Be Present	Cannot be present – cannot be proceed or could cause conflict

Table F.7 RS (Requirements) scale for the BBC's GK Scale concept. This scale is used to define the importance of Metadata elements.

At the European level, the PrestoSpace project [18] considers the use of ontology-based information extraction methods for the automatic semantic classification of audio-visual content.

Annex G Glossary of terms

This annex provides definitions and examples of commonly used terms.

Agent *A software programme that performs information gathering, information filtering, and/or mediation on behalf of a person or entity*

Example

An agent in a hard-disk recorder could automatically record all TV programmes that fit the user's preferences.

Adapter *Software that connects a system component to middleware*

Remarks

Middleware has its own API, which may not be recognised or understood by the system components you want to integrate. This is where adapters are used. Adapters translate between the system's API and the middleware API.

Typically, a vendor of a middleware solution will be able to offer adapters for the most frequently used systems. If not, a specialised adapter for that system has to be written.

Example

A broadcaster wants to integrate its ERP-system into its production middleware. The middleware provider has a standard adapter for the ERP-system in use and can easily interface the two together.

API *Application Programming Interface*
The calling conventions (interface) by which a computer programme communicates with another computer program

Remarks

APIs can be seen as the lists of valid commands that a piece of software allows you to use. APIs are specified in a computer language and are the key to interfacing with the computer programme.

Example

A Newsroom system with a fully, publicly, specified API, increases the chance that other parties can interface with it. It allows developers of, for example, autocue systems, to interface with the Newsroom system by following its API. The API could contain information on: how to retrieve the latest autocue-text, how to notify the Newsroom system in case of a problem, etc.

Asynchronous process *Refers to a process that operates independently of other processes.*

Remarks

Asynchronous processes do not follow a pre-described timing-model

Example

E-mail is a typical asynchronous type of process, as the e-mail sender cannot predict when an answer will be received. He thus does not wait for it (which he would do in a telephone conversation).

Bread and Butter services *Common services (in a specific domain)*

Remarks

In this document we use the term 'Bread and Butter' services for those services that are commonly used by broadcasters (and are not pervasive services). These services are good candidates for standardisation in a broadcast domain model.

Example

Essence transcoding, material stream control, etc.

Broadcast domain model *See domain model.*

Broker *An entity used to exchange data between systems*

Remarks

A broker guarantees the exchange to have a certain integrity.

Example

As a metaphor, think of the stock exchange trader.

Client *A computer system or process that requests a service of another computer system or process.*

Remarks

A client is part of a client-service architecture, where clients ask for information from (central) servers. In the middleware context, a client can be any computer/piece of software that is using services from the middleware layer.

Example

A graphics workstation can be the client of an archive-service providing background maps.

Component *A 'black box' with known and described interfaces*

Remarks

A component is a 'black box' with known and described interfaces, which can be used without knowing anything about its internal structure or internal functionality.

Example

Software can be written as a collection of components. In Object Oriented programming the idea is to purposely make the objects behave as black boxes with known interfaces, so they can be rewritten internally without changing the behaviour of the complete system.

Domain model *A model for a specific business domain*

Remarks

A domain model describes the functional elements used in order to provide solutions *specific* to a business domain, in our case the broadcast domain. It should not describe the internal workings of the elements, but just encapsulate the essential aspects related to the domain.

Example

Currently a broadcast domain model does not exist.

Event-based *Design concept where systems trigger and react to 'events'*

Remarks

Events can, in principle, be issues at any moment in time. This is contrary to a system that, for example, regularly polls another system to see if there were changes.

Example

A hard-disk server might trigger a video switch to switch to the backup before it goes offline. The trigger can be event-based, so it will only trigger when the condition applies.

Framework *The underlying structure of something*

Remarks

The word framework is also used in the context of complex software structures.

Example

For example a well-thought-through software framework typically takes longer to develop than an ad-hoc solution, but it provides a better basis for later extensions to that software.

Integration *A process through which organisations go when there is the need to make systems work together in an orchestrated way*

Integration architecture *The set of strategies and methods that are used to solve system integration problems*

Interface *A shared boundary between two entities (processes, modules, humans, ...) offering a point of communication between the two*

Remarks

The architecture with which to achieve system integration may depend on various aspects, e.g. on your particular business model (products and operations), your particular IT-environment, which may or not be distributed around your organisation, and many others.

Loosely coupled *Refers to an approach to designing interfaces between modules, in a way that reduces the interdependencies between the modules.*

Remarks

Loosely coupled modules are used to limit the risk of changes in or to modules having unwanted/unexpected effects on other parts of the system.

Example

Because the software featured loosely coupled modules, the replacement of a module had minimal impact on the overall system.

Message exchange *Method for exchanging data between computer programs*

Remarks

Message exchange can be compared to computer programmes e-mailing each other. This allows for the creation of distributed applications, where parts of the application run on different computers, interconnected by the network.

Message exchange is asynchronous (not following a specific timing model) and requires the recipients of messages to interpret their meaning and to take appropriate action.

Example

Booking a resource for the Eurovision network using a message exchange system. The resource manager will check the availability after receiving the message and send back a response. In the mean period (which may be very short) you don't know if the resource will be available.

Middleware *Software used for coupling high level system components (applications and user interfaces) with basic system components (network and data storage components)*

Remarks

Middleware translates functionality between the components and allows many different networked systems to be integrated together.

This integration allows the high-level system components to use the basic components (vertical integration). It also allows the high-level system components to integrate with each other (horizontal integration).

Middleware is often referred to as 'software glue' or the 'interoperability layer'.

Typical middleware functionality includes object brokerage, message exchange and directory services.

Example

CORBA middleware between play-out servers and play-out automation, that allows a play-out automation system to interface to many different play-out servers. In case a play-out server fails, a replacement can easily be added to the network, even if it is of a different make.

Object brokerage *Can have different meanings, depends on context*

The technology used to share (distributed) objects

Remarks

An object can be a software component, a software class, a piece of content, a piece of Metadata, a piece of essence, etc. An object should always be uniquely identifiable.

One of the advantages of object brokerage is that the components are automatically recognised by the system. This allows, for example, for the addition or removal of system component, without having to reset or reconfigure the system.

Examples

Searching for the availability of a rendering machine. The object broker provides a list of the rendering services available, allowing the system to provide the user with the most attractive one (e.g. the fastest).

Searching for a piece of content in an archive. The object broker provides a list of all pieces fulfilling the search query and the system presents them to the user.

Pattern *A document that describes a general solution to a design problem that recurs repeatedly in many projects*

Remarks

Software designers adapt the pattern solution to their specific project. Patterns describe a design problem, its proposed solution, and any other factors that might affect the problem or the solution.

Patterns allow for reuse of knowledge and promote the use of common terminology.

There is a rule of thumb that says that a successful pattern should have established itself by leading to a good solution in three previous projects or situations.

Example

An NLE-developer wants to allow the user to select files. He needs this kind of selection many times in different contexts (e.g. selecting project file, selecting files with EDLs, etc.). He describes the commonality as a pattern, which he then reuses in the programming of the software.

Pervasive services *Services not specific to a certain domain (in our case: not specific to broadcasting)*

Remarks

Pervasive services are reusable by other services and applications. The use of pervasive services allows users to benefit from solutions worked out in other industries.

Example

Authentication, session management, workflow management.

Portal *A container for many kinds of content or functions*

Remarks

There are multiple uses of the word 'portal'. It is often used to refer to web-portals, but also for special technology allowing you to integrate various components on a 'portal server'.

Example

Setting up a website to allow the clients of a bank to access their accounts using a Internet-browser.

Real-time *A characteristic of systems which respond to stimuli within a certain time*

Remarks

Used to describe a system that must guarantee a response to an external event within a given time.

'Real-time' is also often used to indicate material transfers at normal playing speed.

We distinguish between three types of real-time related terms:

hard real-time:	event that must happen within a very small delay, typically expressed in milli- or micro-seconds (e.g. within 7 TV-lines)
soft/near real-time:	event may happen after some (small) delay and can typically be implemented as a 'play@time_x' event.
non real-time:	no timing constraints

Example

hard real-time:	a video-switcher that must respond to a button-press within a frame
soft real-time:	a play-out server that must start a programme at an exact moment, but which is known (scheduled) a day in advance
non real-time:	garbage collection (in software)
other use:	A video copy between two servers at a speed of 4 x real-time.

Role *An identifier for behaviour of somebody or something in a specific activity*

Remarks

In Use Cases (a type of modelling) actors appear with a certain role. This is similar to the everyday use of the word role.

Example

In a particular use case, an actor is identified who plays the role of system administrator.

Rule-based *Based on a collection of 'if...then' rules*

Remarks

A rule-based system is one which consists of a collection of 'if...then' type rules. If the condition applies, the rule is executed.

Example

An expert system trying to mimic an operator of a chemical plant.

Servant *A piece of software that actually performs the actions described by an interface definition*

Remarks

The interface basically is only a definition, but the servant is the real code doing the work.

Example

For a storage-service: a file-request servant that returns a file based on the filename it is given.

Service *A system that provides one or more functions of value to the user*

Remarks

In middleware a service typically refers to software (a servant) providing certain functionality.

Example

A directory service allows you to find all users currently logged in.

SOAP *Simple Object Access Protocol*

Remarks

SOAP provides a way to communicate between applications running on different operating systems, with different technologies and programming languages. It runs over HTTP.

Super service *Combination of services in order to perform a more complex task*

Example

The most widely spread type is the category of Web-services.

Synchronous process *Refers to a process that is dependent on other processes*

Remarks

A synchronous process follows a specific timing model. The notion of 'timing' is wide, in the sense that it does not have to mean 'following a heart-beat'. It could also consist of waiting for a message from another process.

Example

File-transfer is a typical synchronous process, where each part of the file is only sent after the previous part is acknowledged by the receiver. In the meantime the sender waits and does not do anything else.

Tightly coupled *Refers to a usually undesirable approach to designing interfaces between modules in a way that features strong interdependencies between the modules.*

Remarks

Tight coupling carries the risk of inflexibility and unexpected effects on other parts of the system in the case of changes to the tightly coupled module(s).

Example

Because the software featured tightly coupled modules, the replacement of a module resulted in the need for much software re-engineering to allow the system to work again.

Transaction *A process consisting of a confirmed request and response. A transaction either completes or fails (is rolled back), but for the user always maintains the system in a deterministic and consistent state.*

Remarks

Transactions basically make sure the system handles requests as expected and does not provide erroneous results or ends up in an erroneous mode.

Rolling back	If you invoke an action which failed, rolling back will ‘undo’ the action to return to the original state.
Deterministic	A process supplied with the same input, provides the same output (‘you know what you get’).
Consistent	Means that when the transaction has been completed it follows the data constraint rules.

Example

A payment transaction to acquire the rights to use a video-fragment from an archive. For the example we assume you pay for it to get access.

Rolling back	If the payment failed, the video-fragment will not be in your possession and your account will not be debited (this was the state you started from).
Deterministic	Every time you acquire the rights, you'll get access to the fragment and your bank account will be debited. It will never happen that you will get the fragment for free or pay for nothing.
Consistent	The correct amount of money is subtracted from your account. Even if the money in reality is subtracted eurocent by eurocent, in the end the complete amount will be subtracted (you will never see an intermediate total on your account).

UDDI *Universal Description, Discovery, and Integration*

Remarks

UDDI is an XML-based registry for business listing on the Internet. Its ultimate goal is to streamline online transactions by enabling companies to find each other on the web and make their systems interoperable for e-commerce. UDDI is often compared to a telephone book's yellow pages. It allows businesses to list themselves by name, product, location, or the web-services they offer. Each company can have an internal UDDI catalogue.

Use case *Use cases are the UML modelling technique for formalising the functional requirements placed on systems.*

Remarks

Use cases are a type of models developed as part of the Unified Modelling Language (UML). Use cases present a part of a system's functionality, as seen from the outside of a system. The description is in the form of a set of messages the system exchanges with (typically) a user of the system (called an actor).

Use cases do not describe the internal behaviour of a system.

See Annex E for a detailed example.

Web-services *Technology including a mechanism explaining how and where to use it*

Remarks

The services could address both data and functionality. A formal description of a web-service allows for access via a totally open and standardised interface. This, in theory, makes integration possible between all services and (other) systems.

Re-use is ensured using catalogues of services. For example: a catalogue of services in your company or public catalogues where companies can decide to share some of their services (see also UDDI).

WSDL *Web Services Description Language*

Remarks

WSDL is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. WSDL is extensible to allow description of endpoints and their messages regardless of what message formats or network protocols are used to communicate. WSDL can be used in conjunction with SOAP, HTTP GET/POST, and MIME, for example.

Annex H References

[1] The Grid: Blueprint for a New Computing Infrastructure, Ian Foster, Carl Kesselman, ISBN 1558604758

[2] Relation between MDA and the Zachmann framework:
http://www.omg.org/bp-corner/bp-files/MDA-Zachman-Framework.pdf

[3] Gartner article on ICC: http://www.ebizq.net/topics/compliance/features/5360.html

[4] Staffing the Integration Competency Centre, Ross Altman, EAI Journal, May 2003

[5] EBU/SMPTE Task Force for Harmonised Standards for the Exchange of Program Material as Bit Streams, "Final Report: Analyses and Results", 1998, available via: http://www.ebu.ch

[6] The Zachmann framework: http://www.zifa.com

[7] Information on the Integrated Definition language: http://www.idef.com

[8] Globus GT2: http://www.globus.org/toolkit/download/index.html

[9] Physiology of Open Grid Services Architecture: http://www.globus.org/research/papers/ogsa.pdf

[10] The anatomy of the Grid: http://www.globus.org/research/papers/anatomy.pdf

[11] NASA's Information Power Grid: http://www.ipg.nasa.gov

[12] Grid application in magnetic fusion research: http://www.globus.org/research/papers/fusion02.pdf

[13] Grid middleware (Legion): http://www.legion.virginia.edu

[14] Grid middleware (Unicore): http://www.unicore.de

[15] Grid-enabled CORBA: http://www.ipg.nasa.gov/workshops/papers/NPSS_CAS_paper.html

[16] Semantic web (W3C): http://www.semanticweb.org

[17] Web Ontology Language (OWL): http://www.w3.org/TR/owl-features/

[18] The PrestoSpace project: http://prestospace.ina.fr

Other relevant information

- The UML modelling language: http://www.uml.org
- Tutorials on UML: http://www.uml.org/#Links-Tutorials
- OMG, Object Management Group: http://www.omg.org
- CORBA services: http://www.omg.org/technology/documents/formal/corbaservices.htm
- Introduction to web-services: http://webservices.xml.com/pub/a/ws/2001/04/04/webservices/
- W3C activities on web-services: http://www.w3.org/2002/ws/
- Examples of middleware products: http://www.ibm.com/middleware/automation/uk
- The File Interchange Handbook, Brad Gilmer, Focal Press, ISBN 0240806050
- The MXF (Material Exchange Format) specification: http://www.smpte.org
- MXF test centre: http://www.irt.de/mxf/

--End of document--