BBC

R&D White Paper

WHP 042

September 2002

The use of component-based technology to support EPG data capture

N. J. Megitt, S. Procter, A. Ruto, D. Bland, P. Layton, C. Mace, R. Penny, M. Robb, D. Waring, N. Yeadon

Research & Development
BRITISH BROADCASTING CORPORATION

BBC Research & Development
White Paper WHP 042

The use of component-based technology to support EPG data capture

N. J. Megitt, S. Procter, A. Ruto, D. Bland, P. Layton, C. Mace, R. Penny,
M. Robb, D. Waring, N. Yeadon

Abstract

This paper describes the requirements of a programme metadata capture system for use by broadcasters in the digital age and presents solutions for implementing some of those requirements. The requirements fall into two categories, system requirements and user requirements. System requirements include expandability, data flexibility, fault tolerance, scalability and upgradeability. User requirements include access control, simultaneous visibility of data changes (referred to as active refresh), and distributed working. The proposed solutions include use of a component standard such as CORBA, with asynchronous communications and loose coupling of components; data self-description in terms of semantic types, format types and structure; and a novel hierarchical schedule representation based on schedule instructions.

This document was originally published in the Conference Publication of the International Broadcasting Convention (IBC 2002) Amsterdam, 12-17 September 2002.

© BBC 2002. All rights reserved.

White Papers are distributed freely on request.

Authorisation of the Chief Scientist is required for publication.

© BBC 2002. All rights reserved. Except as provided below, no part of this document may be reproduced in any material form (including photocopying or storing it in any medium by electronic means) without the prior written permission of BBC Research & Development except in accordance with the provisions of the (UK) Copyright, Designs and Patents Act 1988.

The BBC grants permission to individuals and organisations to make copies of the entire document (including this copyright notice) for their own internal use. No copies of this document may be published, distributed or made available to third parties whether by paper, electronic or other means without the BBC's prior written permission. Where necessary, third parties should be directed to the relevant page on BBC's website at http://www.bbc.co.uk/rd/pubs/whp for a copy of this document.

THE USE OF COMPONENT-BASED TECHNOLOGY TO SUPPORT EPG DATA CAPTURE

N. J. Megitt, S. Procter, A. Ruto, D. Bland, P. Layton, C. Mace, R. Penny, M. Robb, D. Waring, N. Yeadon

BBC Research & Development, United Kingdom

ABSTRACT

This paper describes the requirements of a programme metadata capture system for use by broadcasters in the digital age and presents solutions for implementing some of those requirements. The requirements fall into two categories, system requirements and user requirements. System requirements include expandability, data flexibility, fault tolerance, scalability and upgradeability. User requirements include access control, simultaneous visibility of data changes (referred to as active refresh), and distributed working. The proposed solutions include use of a component standard such as CORBA, with asynchronous communications and loose coupling of components; data self-description in terms of semantic types, format types and structure; and a novel hierarchical schedule representation based on schedule instructions.

INTRODUCTION

One of the consequences of the digital revolution that the broadcasting industry is currently undergoing is a rapid change in requirements for the handling of programme metadata. Rather than being viewed as a fixed 'write-once' business tool, the capture, manipulation, storage and distribution of programme metadata must be treated as an integral part of the programme-making process. The very nature of the programme schedule is changing; modern scheduling includes features such as repeating loops of programmes and internet streaming schedules, and so the old model of a schedule as a simple linear list of items is no longer sufficient. Also, the final destinations of the data may include not only EPGs (Electronic Programme Guides) and Now and Next information, but also interactive TV applications such as the BBC's iBar service (currently available on BBC digital TV). For a system to support EPG data capture on a long term basis, the system must be flexible such that changes in user requirements can be supported and new technologies can be adopted with the minimum amount of system redesign.

Classical software solutions for data capture and manipulation involve a database, network connectivity, and a client capable of editing the data. Implementing a change in a system like this requires changes to all parts of the system, including rolling out new clients to all users. By using a component-based architecture and building in data self-description mechanisms, the number of places in which changes need to be made can be reduced, and new client rollout can be eliminated from commonly occurring change scenarios.

This paper describes work done at BBC Research & Development to investigate the requirements of a system for the capture and manipulation of programme metadata, and to develop a system that meets those requirements.

REQUIREMENTS OF A PROGRAMME METADATA CAPTURE SYSTEM

The purpose of a programme metadata capture system is to collect and organise metadata, allow it to be modified and checked and send the data on to whichever systems need to use the data. The collection may be done by direct user input or import from other systems. The requirements of a system to fulfil this role derive from the functionality required by the system as a whole (system requirements) and that required by each individual user (user requirements). The system requirements and user requirements for a metadata capture system are described below.

System Requirements

- **Data consistency.** A core part of the capture system must be the storage. In this respect it is inevitable that the capture system contains some kind of database. This must obey traditional Relational Database Management System (RDBMS) values such as atomicity. In addition, the capture system as a whole must present the same behaviour as the database. The subject of RDBMS is beyond the scope of this paper, the system requirement of data consistency will not be addressed any further.
- **Expandability.** It is not known at the time of writing what kinds of metadata might be required in the future. Indeed it is expected that the set of *semantic types* of the stored data will grow in time. Therefore it is necessary to be able to update the capture system by adding new semantic types at arbitrary times without requiring major system changes. Note that it may be necessary to restrict the permitted *format types* of such new semantic types or to update client software to be able to present unusual collections of data in ways that make them easier to understand. Nevertheless older pre-update client software should be able to work with a standard set of format types regardless of the semantic types involved.
- **Data Flexibility.** The internal data model used by the system must obviously be sufficiently powerful to handle all the current types and formats of schedule and programme metadata with which it will be presented. It is desirable that the model is also flexible enough to represent other new schedule structures that may emerge as the broadcast environment evolves and develops.
- **Fault tolerance.** The capture system forms a vital link in the broadcast chain. It is a service that may in time be used by other diverse systems. In the same way as a working presentation studio is required at all times during transmission of a broadcast channel the metadata capture system must always be functional. This means that the capture system must be fault tolerant in both hardware and software.
- **Recoverability.** It is impossible to guarantee that any complex computer system will never break. Should the system fail it must be possible to restore it quickly with minimal loss of data. What this means in practice depends on the specific requirements of the broadcaster.
- **Scalability.** The number of users that a system must be able to cope with is not always known at the time of design. In a large organisation, there is a risk that a system might become a “victim of its own success” – the more useful a system is, the more people will want to use it! Scalability is often a difficult attribute to retroactively add to a system; therefore it is important that it is designed in from the beginning.
- **Upgradeability.** When planning a metadata service that will be available at all times it is necessary to plan for major changes to that service, e.g. installing new core software. The simplest mechanism for doing this would be to prepare the changes, shut the service down, implement the changes and start the service up again.

However, most broadcasters need to be able to rely on their systems working without such stoppages so it is a requirement that if possible the system should not need to be completely stopped in order for changes to be made. It is also important to minimise the time required to perform such upgrades.

User Requirements

- **Access control.** It should be possible to specify security privileges for both read and read/write operations on the basis of many different criteria, e.g. user or user group, type of data, and data location. In a service accessed by many different people, each with a different specialisation, it is necessary to be able to specify different security roles for every user. For any given piece of information the system needs to know whether the user concerned is allowed either to view it (read access) or to read and modify it (read/write access). The data must be classified in terms of its semantic type (e.g. programme name, programme genre, start time) and its location.

  The semantic type is related to the user's task set. For example, a user whose job it is to make sure that the programme times are correct will need both read and write access to the time types, as well as being able to read whatever it is that identifies the programme. However it is conceivable that a user whose task it is to make the genre classifications correct would need read and write access to the genre types as well as read-only access to the name and description types.

  The location of a piece of information relates to the structure in which it is placed. If a broadcaster organises its services in days and channels then the location can be specified in terms of these. Some broadcasters transmit different services to different places under the same channel banner. In this case the location must include the region as well.

- **Active refresh.** Users should see other users' changes to the data they are viewing as soon as it happens. A metadata service that contains diverse semantic types of information will need to be viewed by diverse users in multiple locations. In a reactive broadcasting organisation such as the BBC, changes may be happening very close to transmission. It is therefore vital that any changes are visible to all interested users as soon as they have been made. Even if changes are not made close to transmission, there may be more than one user working on closely related data simultaneously; in this case too it is beneficial that each sees the others' changes as soon as they have been committed.

- **Distributed working.** In broadcasting organisations that are diverse and that support diverse semantic types of metadata, it is necessary to allow users in different locations to have access to the service.

MEETING THE REQUIREMENTS OF A PROGRAMME METADATA CAPTURE SYSTEM

This section describes our proposed solutions to some of the aforementioned system and user requirements. The solutions consist of component-based technology, data self-description and a novel flexible schedule model.

Component-Based Technology

Currently, virtually all large-scale, complex software systems are designed in an object-oriented manner. The functionality of any such large system can usefully be divided across a number of interconnected components or objects. By distributing these objects across a computer network, many of the requirements outlined in the previous section can be met. To

support the diverse needs of different parts of a large broadcasting organisation mandates the use of open standards to define the interconnection infrastructure between the various system objects of a distributed architecture. CORBA, the Common Object Request Broker Architecture (1), was designed for exactly this purpose. The use of an Object Request Broker (ORB) hides all low-level platform-dependent communication issues such as byte ordering, making it relatively easy to port system components between platforms with minimal changes to the source code. Although other middleware solutions exist, many of these are proprietary and/or platform specific, limiting the portability of any system that uses them. Other emerging standards have not yet reached the level of maturity required to satisfy the strenuous demands of a broadcast-critical environment. CORBA is the most widely used platform-independent open standard for middleware and is therefore a natural choice for system interoperability within a large organisation.

Some benefits of using an architecture in which the processing is split into different objects connected by CORBA are immediate, providing the individual objects are implemented carefully. Fault tolerance can be provided in software and hardware by running the system on multiple computers connected by a network, each one running a subset of the processes. Monitoring must be built in from the start so that if an individual process stops working correctly it can be restarted, if necessary on a different computer. Of course, the monitoring process itself must be fault-tolerant.

If this automatic process restarting capability is augmented with the ability to monitor the load on each processor in the system, and this information is used to select which processor each process invocation is run on, then load-balancing behaviour can be achieved with very little additional effort. The task is further simplified by the advanced capabilities of CORBA's inter-ORB communication protocol, which make it possible to forward method calls on objects to server implementations regardless of where they are physically located, and to redirect calls as necessary. With this load-balancing functionality, increasing demands on the system can be met simply by increasing the number of available processors, and the requirement for scalability is fulfilled.

In a well-designed distributed system, the system requirements of recoverability and upgradeability should also come naturally as a by-product of fault tolerance, since the system as a whole must continue to function correctly (albeit possibly in a reduced capacity) even when a part of it ceases to function. Therefore a system upgrade simply involves shutting down the process to be upgraded, before starting a new process in its place. Recovering from failure of some part of the system involves a new process correctly starting and continuing from the point when an existing process failed.

Since the processes that comprise the system must be able to function with a high degree of independence, it follows that the communication between them must be asynchronous. Otherwise, if process A fails while process B is waiting for a response from it, process B would become blocked. Any processes dependent on process B would then also block, raising the spectre that the entire system could grind to a halt.

CORBA defines interfaces for certain standard services that are implemented by most ORB vendors. One of these services is known as the *event service*, which provides objects called *event channels*. The essence of an event channel is that it is possible for one or more objects, known as *suppliers*, to place events on the event channel. All *consumer* objects connected to the event channel can then receive these events. This is more than a design abstraction: it is also an implementation abstraction. It means that the problems of asynchronous inter-process communication can largely be set aside.

Event channels also provide a mechanism that can meet the requirement that users see changes to data they are viewing regardless of who made those changes (referred to as active refresh). Any process that provides functionality that enables changes in data to be

committed should supply those changes to an event channel. Any processes that are interested in the data maintained by that process should subscribe to that event channel: they should become consumers of it. Care must be taken with event channels because they are essentially buffers, which can fill up. In this case events may get lost. However, there are other CORBA mechanisms for guaranteeing delivery, such as the *notification service*. Which of these mechanisms is used is an implementation detail.

The user requirement of distributed working necessitates interconnections between different clients and the system. The use of open standards such as CORBA, to define the interconnection infrastructure between the various system objects of a distributed architecture facilitates the distributed working user requirement.

The adoption of a component standard such as CORBA to implement a distributed architecture can facilitate and in some cases meet the system requirements of fault-tolerance, scalability, upgradeability, and recoverability in addition to the user requirements of active refresh and distributed working.

Data Self-Description

The requirement that new semantic types of metadata can be added to the system at any point without requiring new versions of software can be met by describing the semantic types internally within the system rather than stating them externally at design time. The client must know how to deal with certain basic data types, however the set of commonly used data types can be enumerated and this set is fairly small: numbers, times, dates, strings and pick-lists of the above form the backbone of this set. All data processing logic will be kept within the system services rather than the client, and data descriptions will be passed to the client along with the data itself so that the client has sufficient information to provide the required editing functionality. Thus changes to the internal data structures or business logic of the system will not require a new version of the client software.

To achieve this goal, the metadata capture system needs to maintain a description of its own data: metadata about the metadata! This approach has implications for the storage of the data. In a traditional relational database, the database administrator maintains control of both the semantic types and the data types and allocates appropriate storage to them. What is appropriate depends on statistics relating to the actual values of the data. The decisions made by the administrator have major performance implications for the system as a whole. Therefore they cannot be done away with simply because we have internalised the description of the data. Rather, the self-description should contain storage information as determined by the administrator. Since the semantic type information is related to the structure it is possible to allocate different storage parameters (e.g. table and field names) to the same data types and semantic types in different locations. This gives greater control to the administrator than would normally be available. Maintaining the self-description is not seen as a user role but as an administration role.

This data self-description provides an ideal platform for implementing the user requirement that access control be provided on the basis of data type and/or location. Every semantic type and location is described so it is a simple matter to assign to each one a security role. Every system user has a set of security roles defining the data which they are permitted to read and/or modify. In this scheme, the ownership of any given piece of data is not seen as relevant to its security; the organisation as a whole is the owner.

A Flexible Schedule Model

Traditionally, a programme schedule has been thought of as a simple linear time-ordered list of programmes. However, this definition is far too limited to cover the wide range of new

service formats that are beginning to appear with the advent of digital broadcasting and the convergence of digital media. Schedules can include structures such as repeating loops of programmes, opt-out programmes on the basis of local region or platform, and partial schedules inherited from other services. The schedule model must cope with demands that result from other broadcasting considerations such as the need for a number of services to share the same broadcast bandwidth, for example on a time-exclusive basis. This requirement relies on enforcing the condition that only one of a set of services can be on-air at any given instant in time. Also, schedules need not follow the traditional time-ordered model at all, for example in the case of on-demand services. In order to model all these possible schedule structures, and other potentially unforeseen variants which may be used in the future, a rich, powerful and extensible schedule modelling language is required.

A potential solution to this requirement is the concept of a *schedule instruction*. The idea behind this concept is that all types of information pertaining to a schedule, for example information about services, distribution platforms and regional availability as well as programme information such as name, description, timeslot, genre etc., can be contained within a schedule instruction. Schedule instructions can also contain other schedule instructions, allowing schedule information to be represented by a hierarchical tree of schedule instructions. This hierarchical structure is an example of the 'Composite' design pattern (2). An example of such a hierarchical schedule depicting part of the BBC1 service is shown in Figure 1.

This hierarchical schedule modelling approach is ideally suited to the task of representing schedules with regional structure. For example, many services consist of a number of programmes which are common to a large geographical area such as the UK, along with a smaller number of regional opt-outs which may exist at the level of nations (for example, a sporting event available only to viewers in Scotland) or smaller regions (local news, weather, or traffic reports). Other structures can also be modelled using instructions that modify the meaning of their descendants, for example a loop instruction could be defined to signify that the programmes defined by its descendants are repeated over a number of iterations. The flexibility of this approach stems from the fact that new schedule instructions can be defined to cope with any new schedule modelling requirements that may arise, without having to redesign the entire modelling language. The use of a flexible schedule model to represent the programme schedule fulfils the system requirement of data flexibility.

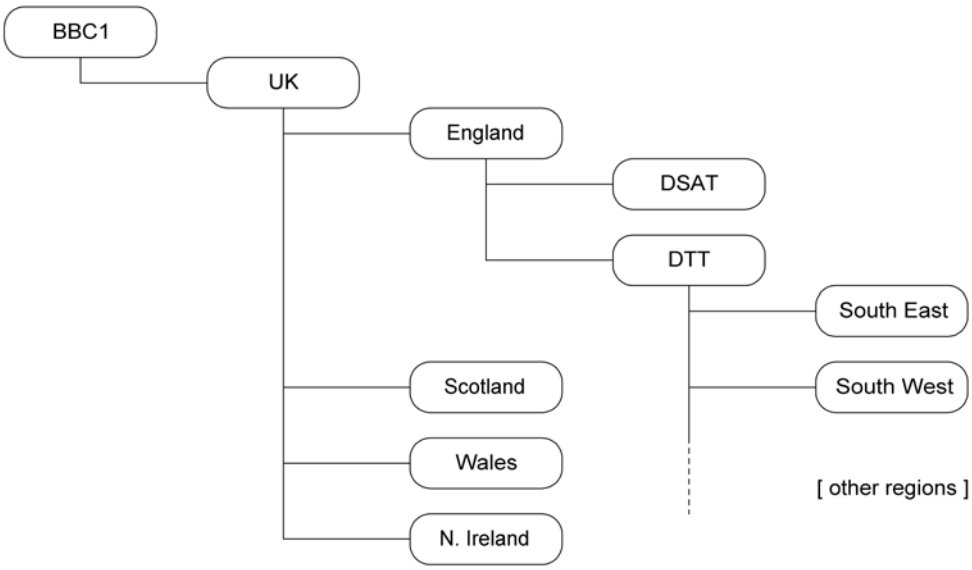

Figure 1 - Hierarchical Schedule Representation

IMPLEMENTING A PROGRAMME METADATA CAPTURE SYSTEM

The BBC's SID project is currently working on the implementation of a programme metadata capture system as described in this paper. It is a layered design whose four layers are persistence, system services, session management and clients. A prototype system has been demonstrated which takes raw schedule information from other systems within the BBC and stores it in a relational database. A distributed middleware layer comprising a number of load-balanced schedule servers plus ancillary services such as session management allows PC-based GUI clients to view and modify the schedule information. The prototype system has demonstrated the use of a distributed component-based architecture to provide fault-tolerance and scalability and to aid distributed working; a novel hierarchical schedule model to provide flexibility of data representation; and data self-description techniques to aid expandability, and as a method of implementing fine-grained access control.

FUTURE DEVELOPMENTS

There are numerous ways in which a system as described may be developed further. Many of the principles described apply not only to metadata capture systems but also to other data capture applications. Defining a minimal set of semantic and format types will help to permit the system to communicate with other metadata systems. Another aid to such intercommunication would be the establishment of standard data formats and interfaces.

CONCLUSION

This paper has described a subset of the system requirements and user requirements of a metadata capture system and presented solutions to those requirements based on the use of a component-based technology, a data self-description mechanism, and flexible schedule model providing a hierarchical schedule representation based on schedule instructions.

The adoption of a component standard such as CORBA to implement a distributed architecture can facilitate and is some cases meet the system requirements of fault-tolerance, scalability and upgradeability, in addition to the user requirement of active refresh. Data self-description is one solution to implementing the user requirement of access control and aids the system requirement of expandability. The use of a flexible schedule model to represent the programme schedule fulfils the system requirement of data flexibility.

REFERENCES

1. Object Management Group. *Common Object Request Broker Architecture (CORBA/IIOP)*. Available from http://www.omg.org/.

2. Gamma E., Helm R., Johnson R. and Vlissides J. *Design Patterns – Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994. ISBN 0-201-63361-2.