An Open Interface to a DTV Data Server

Dinkar Bhat, Dave Catapano, Brad Holcombe, Ilya Shnayder, Gomer Thomas
Triveni Digital, Inc.
Princeton Junction, NJ

Abstract

DTV data broadcasting offers broadcasters new revenue sources that can offset the infrastructure costs in making the transition to digital broadcasting.

At the head-end, the two major players are the content providers and the broadcasters. They should be supported by two distinct components in a data broadcast system architecture, since they have unique roles. Information such as bandwidth allocations, content schedules, and content itself has to be passed between the two components. The communications interface for passing this information must be very flexible, since this information could significantly differ between applications.

This paper describes an open, flexible interface for communication between the two components.

DTV Data Broadcasting

In an ATSC DTV broadcast stream, a fixed bandwidth of 19.39 Mb/s is available. Part of the stream is used for video and audio packets. Data packets can occupy some or all of the remaining bandwidth. The amount of bandwidth available for data changes continuously as the channel lineup and the characteristics of the video and audio vary. It typically ranges between 1.4 and 16.4 megabits/second.

Data broadcast applications can be classified along different semi-orthogonal axes, for example:

- Type of data recipient: An enterprise could target data broadcast towards consumers, or to enterprises. Distribution of real estate listings to local real estate offices would be an example of the latter, while sending personalized content like electronic magazine subscriptions would be example of the former.
- Degree of coupling to a television program: Tightly coupled data would be used to enhance a television program. On the other hand, uncoupled data like software updates would be totally unrelated to any program, and would be transmitted in separate channels.
- Type of Content: Content to be transmitted could be in the form of files like web pages, or streaming content like video or audio.

Data Broadcast Architecture

As described in a previous paper, there are typically three roles in a DTV data broadcast application, namely,

- Content/Service Provider who wants content to be broadcast,
- Broadcaster who provides the broadcast stream containing the content,
- Content Recipient who will use the content.

A DTV data broadcast system should ideally contain three distinct components to meet the needs of the three roles:

- A Data Specifier for the Content Provider, that allows the user to enter detailed specification of the items to be broadcast, including their broadcast schedule and parameters such as the bandwidth to be allocated for different items, the active period over which they must be broadcast, the frequency with which they must be rebroadcast over the active period, etc. The component can also manage the transfer of data to the broadcast station. It will often be
customized to meet the needs of a particular application.

- A Data Server for the Broadcaster, that allows the Broadcaster to allocate bandwidth across multiple content providers, inject data into the broadcast stream, measure bandwidth actually used and generate billing reports periodically. The injection of data follows the schedule(s) specified by the content providers, subject to bandwidth allocations set by the broadcaster and the actual bandwidth available in the stream.

- A Data Receiver for the Content Recipient that extracts data from the broadcast stream, and delivers it to the appropriate application.

In this three-component architecture, it is necessary to define an interface between the Data Specifier and the Data Server, so that schedules and content can be sent from Data Specifier to Data Server, and bandwidth allocations can be sent from Data Server to Data Specifier.

The above architecture can easily be extended to the case where a group of broadcasting stations forms a larger data broadcast network. Content providers at the network level get bandwidth allocations and provide schedules and content to a central operations center. These network level schedules and content are then sent to the individual stations, where they may be combined with schedules and content from local content providers. The broadcast at each station may consist of a mix of network and local content. (See Figure 1.)

### Communications

Here, we discuss three typical types of broadcast applications where the scheduling information to be communicated between the Data Specifier and Data Server would vary:

- **Uncoupled Files**: The Data Server transmits individual files or directories according to schedules provided by the Data Specifier. Here the data is not coupled to any television program, and hence is carried in a data-only virtual channel. An example would be distributing real estate listings from a multiple listing service to individual real estate offices. A schedule would include one or more time slots for file transmission. It could also include parameters for each time slot like the number of times the file or directory has to be re-transmitted (the carousel count) and the interval between successive transmissions. The files and the directories to be broadcast may either be transferred to a cache at the data server, or links in the form of URLs may be given that can be used by the data server to locate content.

  Data transfer to the Data Server can either take place in the “push” mode or the “pull” mode. In the former mode, data can be pushed from the source whenever it is created or updated. In the latter mode, data is pulled from the source at prescribed intervals.

  Additionally, the Data Specifier can specify the mime-types for data that would enable the receiver to automatically launch the appropriate application to use the data. For instance, if a web site with HTML pages is broadcast, and if the receiver is informed about the type of data, then it could automatically launch a browser on receiving the content.

- **Tightly-Coupled Enhanced TV**: The Data Server broadcasts enhancements, so that interactive applications can be enabled at the receiver. The enhancements are tightly coupled to a television program, and are therefore carried in the same virtual channel as the audio/video streams that they enhance.

  There are two primary modes of enhanced TV. In one mode “triggers” are put into the
broadcast stream telling the receiver when to retrieve and display the enhancements, but the enhancement content is obtained via a “back channel” from a remote server. In the other mode both triggers and enhancement content are delivered in the TV broadcast, so the receiver does not need a back channel. The former mode is typically used in analog TV and the latter mode in digital TV.

In the ATVEF interactive TV specification\[^{2}\], the first mode is called Type A transport, and the second mode is called Type B transport.

In the mode where both triggers and content are delivered in the broadcast stream, the output from an enhancement authoring tool consists of the content files and the trigger timings. A specialized Data Specifier must analyze the links among the content items, the bandwidth allocated for the enhancement, the amount of cache available in the receivers, and various other factors,\[^{3}\] and determine the actual broadcast schedule for the content – during what portion of the program each content item must be broadcast, and how frequently or at what bandwidth level it must be broadcast during that portion. Determining the schedule for broadcasting the triggers is straightforward, of course, since that comes directly from the authoring step, and they use a negligible amount of bandwidth in a DTV broadcast.

The Data Specifier must convey enhancement content, triggers, and the schedule to the Data Server. In the case of the triggers especially, the timing must often be very precise. Such a precise schedule would not usually be required for the previous application.

- **Streaming Media**: The Data Server encodes and transmits IP datagram packets that it receives from a streaming media server. The streaming media server could be located remotely from the data server, and could be running at varying schedules over the day, delivering packets of varying sizes, and changing bit-rate.

The Data Specifier has to convey the IP address(es) on which the data server would receive packets from the streaming server. The provider may specify the bit-rate at which the streaming media packets will arrive, primarily for bandwidth management purposes. (For the application to work properly, the Data Server must usually transmit the data at whatever rate it arrives.) In this application it is not meaningful to specify repetition counts or transmit intervals.

Unlike the earlier applications, data has to be pushed from the source to the data server, in the form of IP packets.

A variation on this application is the case when a media server operating under the control of the Data Server is generating the media stream from a file. In this case the Data Specifier may specify the bit-rate, repetition interval, and repetition count to be used by the Data Server in controlling the media server.

Note that a single Data Specifier may schedule combinations of both streaming and file content.

Figure 2 illustrates a likely scenario where one Data Server communicates with different types of Data Specifiers, each handling a different class of application. Thus, any standard interface between Data Servers and Data Specifiers must have the flexibility to handle diverse types of applications, and must be extensible to allow for new applications that may arise in the future.

![Figure 2: Typical Datacasting Scenario](image)

Moreover, it is highly desirable for the interface to be open because:
• It enables custom Data Specifiers to be developed for particular applications by content providers or third party vendors.
• It allows Data Servers and Data Specifiers from multiple different vendors to interoperate seamlessly.
• It enables reuse of software modules between different vendor-supplied software.

XML[4] provides an excellent basis for such an open interface because of its flexibility and extensibility. Moreover, because of its popularity, it enjoys a large expertise-base amongst software developers, and there are many standard software libraries for working with it. Appendix A provides a very brief overview of XML.

Open Interface

This section describes an open interface that meets the needs mentioned above. The Data Specifier and Data Server operate as client and server, respectively. The CORBA/IIOP distributed object architecture[5] is used for the communications infrastructure, because of its openness and its support on a wide variety of platforms. A very brief overview of CORBA is given in appendix A.

Each content provider can conceptually be viewed as an “account” at the Data Server, since the content provider typically contracts for bandwidth to be utilized for data transmission. Properties regarding the account, like contact information and bandwidth allocation profile, can be expressed in XML.

The content to be broadcast is organized into hierarchical structures called catalogs. A catalog consists of products and product groups. For instance, if music is the content, then the set of all albums from classical composers can be seen as one product group, the set of albums from Mozart can be seen as a subgroup of this, and the individual albums of Mozart can each be viewed as a product. Another product group could be the set of albums from rock musicians. The product groups can be nested to an arbitrary depth. Various attributes can be defined for catalogs, groups, and products, including the attributes that define the broadcast scheduling. Such a catalog can be expressed conveniently using XML.

Attributes for a catalog would include the catalog name, version number, date last modified, account to which it belongs, and the list of Data Servers that are to broadcast the content described in the catalog.

Groups are primarily just mechanisms for clustering content, so the primary attribute for a product group is just the group name.

Attributes for a product would include the product name, media type (file, directory, stream, …), data type (MIME type), location (URL for files or directories, source and destination IP addresses for streaming products), product size, flag indicating whether it should be auto-launched on arrival at the receiver, a set of broadcast schedules (with start time, duration, bit-rate, carousel interval, and carousel count for each), etc.

In the case of an interactive TV application, each enhancement could be a product group containing products corresponding to the triggers and the individual items of enhancement content. Each product would have its own schedule for broadcast.

The list of methods that a Data Specifier can invoke, along with their parameters, is specified in a CORBA IDL (Interface Definition Language) file. The parameters to the methods can be XML structures. For instance, if a Data Specifier invokes a method to add a new schedule, then the list of parameters would include a catalog.

The set of methods that the Data Specifier may invoke as a client includes:

• Create a new account: An XML structure giving the account information would be a parameter.
• Delete an existing account: The account identifier would be the parameter.
• Update information about a specified account: An XML structure giving updated account information would be a parameter.
• Add a catalog to an existing account: Parameters would include the account identifier and an XML representation of the catalog.
• Update a catalog in an existing account: Parameters would include the account identifier and an updated catalog.

• Informational methods that return specified account information, or the list of catalogs in a specified account.

All the above methods would be synchronous calls in the sense that the Data Specifier would block until the operation is completed.

In addition, the Data Server can generate events that the client can listen for:

• Events reporting data transfer progress.

• Status events that report the current properties of the data server.

Appendix C lists sample XML Document Type Definitions (DTD) for an abbreviated catalog. It also includes a sample catalog that conforms to the DTDs.

Experiences and Future Plans

The described interface has been implemented in the Triveni Digital SkyScraper™ data broadcast system. It has been used with file-based uncoupled data applications in several field trials with broadcasting stations. The catalogs in these applications contained products for web-pages, software updates, etc. It has also been used for a streaming media application in which a popular streaming video server was generating IP packet streams. Trials are planned for the near future with catalogs containing both file and streaming products.

At the NATPE 2001 show in January this open interface was used with catalogs for interactive TV based on ATVEF. The receiver was a LG set-top box containing an ATVEF-compatible client. Interactive content and triggers were provided by a third-party interactive TV authoring tool.

It is probably too early at this point to define a standard based on this interface, partly because more field experience is needed to refine it, and partly because the system architecture on which it is based is not yet widely supported in the industry. As more broadcasters transition into digital TV, and as data broadcasting becomes more prevalent, standardization of an interface of this sort between head-end components may become very important. Triveni Digital is likely to propose a standard interface based on the concepts in this paper when that time comes.

Summary

We discussed three different classes of data broadcast applications with wide variations in the kind of information to be passed between the two head-end data broadcast components, namely the Data Specifier and the Data Server. We discussed an open interface based on XML and CORBA, and we described the properties of the XML specification. Standardization of such an interface may become extremely important as DTV data broadcast applications become more prevalent.

References


Appendix A

XML is a text-based markup language that is widely being used for data interchange between applications. The World Wide Web Consortium (W3C) is responsible for standardizing XML. As with HTML, data is identified using tags (identifiers enclosed in angle brackets, like this: <...>). Collectively, the tags are known as "markup".

But unlike HTML, XML tags tell the meaning of data, rather than how to display it. For instance, while an HTML tag says something like "display this data in bold font" (<b>...</b>), an XML tag puts a label on a piece of data that identifies it (for example: <message>...</message>).
Tags can also have attributes that give additional information about the tag. Since tags can be nested arbitrarily, XML is ideally suited to represent data hierarchically.

An example XML structure would be:

```xml
<person ssn="123-45-6789">
  <name>
    <first>John</first>
    <last>Smith</last>
  </name>
</person>
```

Here there are four tags ("person", "name", "first", "last"). "ssn" is an attribute of the tag "person". The tags "first" and "last" are nested within the "name" tag.

Although one is free to invent tags freely for structuring data, when different applications plan to interface using XML, it becomes convenient to use a Document Type Declaration (DTD), which specifies the kind of tags and attributes, and their arrangement.

Several good sources of information for XML exist, for instance [4].

**Appendix B**

The Common Object Request Broker Architecture (CORBA) is a distributed object architecture developed by the Object Management Group (OMG) to facilitate development of reliable distributed applications. CORBA provides the communication architecture for a heterogeneous distributed collection of communicating objects.

CORBA defines an Interface Definition Language (IDL) that is used to specify the interfaces (methods) between two communicating objects. An important feature of CORBA IDL is that it is independent of the implementation language of the objects, so it allows implementations in different programming environments to interoperate. OMG has defined bindings of the IDL to various popular programming languages.

**Appendix C**

The following is an abbreviated DTD for a catalog:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!-- Abbreviated DTD for catalog -->
<!ELEMENT Catalog (Group+)>
<!ATTLIST Catalog id ID #REQUIRED
  name CDATA #REQUIRED
  version CDATA #REQUIRED
  date CDATA #IMPLIED
>
<!ELEMENT Group (Product+)>
<!ATTLIST Group id ID #REQUIRED
  name CDATA #REQUIRED
>
<!ELEMENT Product EMPTY>
<!ATTLIST Product id ID #REQUIRED
  version CDATA #REQUIRED
  name CDATA #REQUIRED
  type CDATA #IMPLIED
  source CDATA #REQUIRED
  destination CDATA #IMPLIED
  encodingType CDATA #IMPLIED
  size CDATA #IMPLIED
  startXmit CDATA #IMPLIED
  endXmit CDATA #IMPLIED
  xmitCount CDATA #IMPLIED
  xmitInterval CDATA #IMPLIED
>
```

The DTD implies that the catalog has one or more product groups and each group has one or more products. The list of required attributes for the product group, and the product tags are constrained by the DTD.

In the "Product" tag, the "startXmit" and "endXmit" attributes specify the active period for transmission, the "xmitCount" specifies the number of times the product has to be sent, and "xmitInterval" indicates the frequency.
Appendix D

A sample XML message that conforms to the above DTD is shown below.

<!-- Sample Catalog XML file -->
<!-- 8/29/00 10:34 AM -->

<Catalog id="967559120530"
    name="Music Catalog"
    version="1.0"
    date="08/29/00 10:34:36"
    maxBitRate="1000000">
    <Group id="967559126030"
        name="Music Records">
        <Product id="967559126470"
            version="1.0"
            name="Zepellin Songs"
            encodingType="http"
            type="directory"
            startXmit="11/10/2000 10:00"
            endXmit="11/10/2000 12:00"
            xmitInterval="00:00:15"
            size="2154239" />
        <Product id="967559129470"
            version="1.0"
            name="Abba Songs"
            encodingType="http"
            type="directory"
            source="/Abba Songs"
            startXmit="10/11/2001 13:00"
            endXmit="10/11/2001 15:00"
            xmitInterval="00:00:30"
            size="2181237" />
    </Group>
</Catalog>