Multimedia Customisation Using an Event Notification Protocol

Ricky Robinson, Andry Rakotonirainy
School of Information Technology and Electrical Engineering
The University of Queensland, Brisbane, QLD 4072, Australia
{ricky, andry}@itee.uq.edu.au

Abstract

Online personalisation is of great interest to companies. Event notification systems are becoming more and more popular as a natural candidate to provide personalised services. Although event notification protocols do not immediately spring to mind as the most sensible transport of real-time streams, our approach does utilise a content-based event notification protocol. In this paper we present an architecture able to correlate and filter real-time multimedia streams using an event notification protocol (Elvin) and the Real-Time Transport Protocol (RTP). We demonstrate, through simple examples, how a MPEG-2 stream can be customised to the user in real-time based on his or her subscription. Such an architecture can serve as enabling technology to integrate, correlate and abstract different sources of information such as discrete and continuous events.

1. Introduction

Today there exist some applications that allow users to customise text based feeds to their interests. For example, there are web sites that send regular updates and news via electronic mail to their subscribers. Upon subscription, users can select the subject categories they are interested in, and any e-mail sent to them will contain only information relevant to that user. There are also applications that *scrape* news web sites such as CNN and Slashdot and scroll the news items on a ticker-tape interface as the stories appear on the web site. This paper describes an architecture in which real-time multimedia feeds, specifically MPEG-2 streams, can be customised in a similar way using the Elvin [14] content-based routing protocol. It also shows how correlation can be used to further enhance the user's experience and provide them with more choices.

Section 2 describes two motivating examples, Section 3 summarises related works. This is followed by an overview of the existing technologies that we combined for use in this architecture. Finally, Section 5 describes the proposed architecture in detail and Section 6 gives some examples of how a client application can interface with the architecture.

2. Motivating Examples

This section provides several diverse examples of the way in which a client may use this architecture. Although the examples differ somewhat, they all use the same simple programming interface. Implementations of these examples are provided in Section 6.

2.1. The User Knows the Channel

In this, the simplest use case, the user knows the name of the TV channel he or she wishes to view. This corresponds to the way in which TV and other resources are used today: a person knows there is football on Channel 4 at the current time, so they just turn on Channel 4. In this situation, the only benefit this architecture provides over and above existing systems is that a TV channel may be relayed using Elvin from several redundant Content Servers. If one of the servers crashes, the client will transparently start receiving packets from another server. There would be no reconnection required on the user end. There is also the benefit of being able to identify a content source by name (for example "DW-TV") instead of by an IP address and a pair of port numbers.

2.2. Quick! Turn on Channel 9!

Sometimes, if there's something really extraordinary on television, a friend might call you up and exclaim "Turn on Channel 9! There's something you really need to see!" This scenario allows you to specify that a selection of your friends may automatically display a TV channel on your computer. This is akin to the X Windows functionality of allowing certain hosts to display windows on your desktop (using the xhost command).

This final scenario shows how to correlate multimedia content with notifications from any arbitrary Elvin based application. In this example, the channel name from the MPEG teletext notifications is correlated with an Elvin based chat application. The result is that your friends or colleagues may turn on the TV to a specified channel so that you don't miss the "something you really need to see!".

There are limitless other possibilities, most of which should require no additional programming work.

3. Related Work

There are numerous content-based event systems (Gryphon [17], iBus [9], JEDI [5], Keryx [8], Siena [4],

CORBA notification Service [12], Java Message Service [11]). To our knowledge, nobody has attempted to use any of these event based architectures to distribute multimedia content. More to the point, we are not aware of any attempt to combine event services, event correlators and real-time streaming protocols to deliver customised multimedia content to the end user. It is our view that event based systems and their respective correlation engines provide an ideal platform upon which to build a customised multimedia delivery system.

The following paragraphs review previous projects that attempt to deliver customised media streams. Although none of them marry multimedia with event notification systems, they do provide ideas that proved to be useful in our architecture.

The Medusa project [18] from ORL is an ATM based platform that can support the storage and streaming of multimedia data. One project that took advantage of this platform was a combined ORL and Cambridge University effort to store news broadcasts for later retrieval by users [3]. The key to this architecture is the storage of teletext in parallel with the video and audio tracks. Users can formulate a text query which is then matched with the teletext. Matches are ranked in order of relevance. The user can then retrieve one or more of the matching video streams from storage.

A large portion of the Cambridge project was concerned with an analysis of the relevance of the matches returned to the user. Although the results of these experiments cannot be directly utilised in the architecture presented in this paper due to differing goals, one conclusion does prove useful: teletext serves well as meta-data. That is, in general, teletext provides a good indication of the content of the accompanying video and audio tracks.

The PRISM architecture [1] from AT&T is a system for content discovery. PRISM does not specify how media items are selected by the user, but it does specify how these items are named, and how the ”best” media server is selected from a set. PRISM uses a late binding mechanism, such that users may choose content using a content naming URI, without having to know which content server will serve that content.

The architecture presented in this paper uses ideas from both the Cambridge project and PRISM. It treats teletext as meta-data that describes the content of the video and audio streams, and allows users to select content based on the teletext. However, it also provides location transparency in a similar way to PRISM. The major innovation that this paper hopes to provide is the ability to customise media dynamically. Unlike the Cambridge project and other similar projects, our design does not require that a media broadcast be stored to disk, then indexed, then searched or browsed. Rather, it provides a way in which *live* media can be customised to the users' needs.

4. Components Used in the Architecture

Our architecture will be presented in Section 5. Our architecture combines several components shown in Figure 1. They are a notification service called Elvin, a correlation service, RTP (Real-Time Transport Protocol) and DVB/MPEG2 streams and decoder. This section gives a brief overview of these components.

4.1. Elvin

Elvin [14] is a content based routing service. Consumers subscribe to receive events whose content matches some expression. Producers emit events. The Elvin routing service performs the matching of subscriptions to events, and forwards these events to the relevant consumers. Elvin is anonymous in so far as a producer does not know who is consuming its events, and a consumer does not know who produces the events it receives. An Elvin notification consists of a set of key-value pairs. The keys are strings, and the value may be an integer, float, string or opaque. A subscription expression, sent by a consumer to the Elvin router, controls which events will be received by the consumer.

Elvin has several benefits besides the obvious one of allowing packets to be routed based on the content they contain.

Another benefit is the potential to provide transparent fault tolerance. A publisher of information may have established two or more redundant servers. If the primary server stops transmitting due to a disk crash or other failure, one of the backup servers begins transmitting instead. The client receiving the data will be oblivious to the failure. No reconnection is necessary on the part of the client.

As will be shown in Section 6, the use of Elvin in conjunction with the Elvin Correlator (described below) allows the construction of usage scenarios that were not envisaged at application development time. In other words, even non-technical users can extend the original application by providing simple inputs to a graphical user interface, for example.

4.2. Elvin Correlation

The Elvin Correlation engine [6] draws on techniques used in GEM [10] and other event based systems. Event correlation allows higher order events to be produced from a set of lower order events. The arrival of several lower order events (possibly within a restricted time-frame) might trigger the emission of a higher order event.

An Elvin correlation specification is expressed in an XML [2] dialect. A correlation specification is a rule that consists of Events and Relationships between events, followed by a corresponding action to carry out. Events are either received from a *service* with a well defined name (an Elvin producer), or they are temporal events that represent a specific local time.

Finally, there are a number of actions that may be taken once a rule has been matched. An action may enable or dis-

able a rule or a portion of this rule. It may emit a notification. Or it may execute a call-back function.

The Elvin Correlation service exists as a standard Elvin producer and consumer. The correlation service subscribes to receive component events as they appear in the event part of the specification, and may emit a higher order event depending on the action section of the specification.

Section 6 shows an example of an Elvin correlation specification.

4.3. Real-Time Transport Protocol

The Real-Time Transport Protocol (RTP) is described in RFC 1889 [13]. It is the Internet standard protocol for the transport of real-time data such as live video and audio. RTP has a data channel and a control channel. The control channel is known as RTCP (Real-Time Control Protocol). RTP is transport independent. That is, it may carried over TCP, UDP or any other protocol for that matter. In most circumstances, UDP is chosen because it is more suitable for real-time data than TCP (one doesn't want to delay the playback of a video midstream while dropped packets are re-sent by the transport). A media source might contain many different tracks of data. For example, a Quicktime movie contains both a video component and an audio component. RTP delivers each of these streams in separate *sessions*. This enables more flexibility, since the user may choose not to have the video component delivered. Each session has a data channel and a control channel. When used in conjunction with UDP or TCP, these two channels are differentiated using different port numbers.

4.4. DVB/MPEG-2

MPEG-2 (Moving Picture Expert Group) encodes audio, video and data into a form suitable for storage and transmission. DVB (Digital Video Broadcasting) is a broadcast technology based on MPEG-2 [16]. DVB is a point to multi-point data delivery mechanism with guaranteed quality of services. MPEG-2 is the source encoding of audio and video data that is transmitted over DVB. Data is organised in a manner optimum for broadcast media by using the rigid timing framework of the television signal. Teletext data are conveyed in Packetized Elementary Stream (PES) packets which are carried by the Transport Stream (TS).

DVB/MPEG-2 is the source format for multimedia data in this project. It was chosen simply because it is the most popular mechanism for broadcasting digital television.

The relationship between MPEG-2 and RTP is described in RFC 2250 [7]. This IETF document describes how MPEG-2 is carried by RTP. The audio and video tracks from the MPEG-2 source may be multiplexed and carried in a single RTP session, or they may be separated and sent in their own RTP sessions.

5. Architecture

We assume that subscribers know how to use/interpret the sequences of discrete and continuous event they are requesting.

5.1. Overview

The purpose of this paper is to describe a way in which multimedia may be customised to the end users' needs. Therefore the design is focused on specifying how Elvin, Elvin correlation and RTP can be combined to create customised media feeds. For completeness, we also describe how multimedia can be received from our chosen source (DVB), decoded into its component MPEG-2 Elementary Streams and then sent via ElvinRTP to the end users. Figure 1 shows a high level overview of the proposed architecture. The arrows indicate the *logical* flow of data from the Content Server to the client's end system. The actual path taken by the data is somewhat different.

Teletext [15] is one of the key components of this architecture, for it acts as meta-data that describes the content of the video and audio streams. So the teletext provides the means by which multimedia content can be customised to the user's preference. Depending on the customisation requirements of the user, the teletext notifications from one channel can be correlated with teletext from another channel, or the teletext can be correlated with notifications from a totally separate application. In the simplest of cases no correlation is required. Use cases corresponding to the motivating examples are given in Section 6.

5.2. The Content Server/TIU

Using the terminology of RFC 1889, the Content Server can be thought of as a Transmitting Interworking Unit (TIU). That is, it receives data from a multimedia source (in this case a DVB broadcast), and transmits using RTP. The TIU receives a multiplexed MPEG-2 Transport Stream from a DVB broadcaster. A DVB-S transponder can transmit up to 8 separate Standard Definition programs or 2 HDTV programmes. Each programme may contain video, audio and other data elementary streams. This project utilises only those elementary streams containing video, audio and teletext data. A DVB decoder card or software in the TIU decodes the Transport Stream into its component Elementary Streams. Elementary Streams containing video or audio are sent via ElvinRTP, whilst the text from teletext Elementary Streams is pulled out of the PES packets and sent directly via Elvin as strings (the tel2str module in Figure 1). The teletext notifications also contain the name of the channel to which they correspond.

5.3. The Correlator

The task of the Elvin Correlator is to receive a correlation specification from a client, and *execute* the specification. In this architecture, the correlator will be given a specification that contains the rules for performing some kind of customisation for the user. The specifications will generally contain rules for correlating multiple TV channels, or correlating TV channels with events from other Elvin based

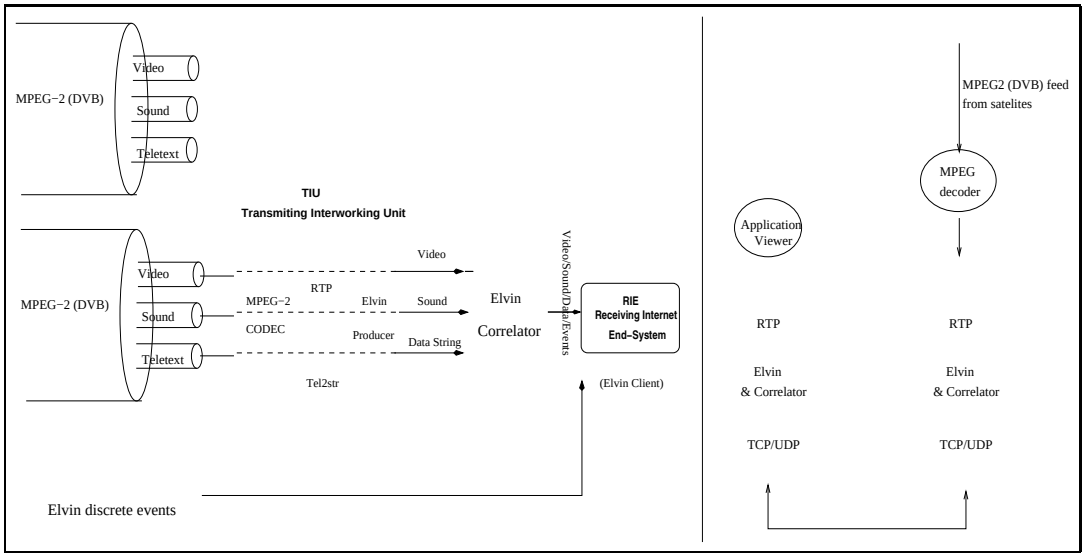

Figure 1. Data flow and the ElvinRTP protocol stack

applications. In this way, the client can create highly customised usage scenarios that were not originally envisaged by the client application developer. At this point, correlation is performed on the teletext notifications. At some later time, it may be possible to perform correlation on the video and audio streams themselves.

5.4. Elvin RTP

Whilst it is not critical in this architecture that Elvin is used as the underlying transport for RTP, it does have its benefits. That is to say, while it is necessary for the teletext to be carried by Elvin so that content customisation may be performed, the audio and video itself could be transported by any suitable real-time protocol such as the Real-Time Streaming Protocol (RTSP) or RTP over UDP. However, we have chosen to transport the audio and video data using RTP over Elvin (ElvinRTP). As stated in Section 4.1, there are many benefits that Elvin provides for our architecture. Among these benefits are location transparency and meaningful transport addresses. As Elvin matures, it may also become possible to have the Elvin router filter multimedia payloads directly, without relying on the teletext component.

Elvin itself can run over TCP and UDP among other transports. For the purposes of ElvinRTP, it is most sensible to use UDP, since we would rather risk losing a few packets than for the entire stream to be held up waiting for a lost packet to be re-sent. The box on the right of Figure 1 shows the protocol stack used in this architecture.

ElvinRTP transport addresses are somewhat different to those used in TCP or UDP. Instead of using an IP address to identify a receiver (or a group of receivers in the case of a multicast address) in conjunction with two port numbers, ElvinRTP addresses are a combination of an Elvin URL of the form elvin://elvin.server.com and a session name such as ”DW-TV”. The data stream is given a notification type of ”DATA” and the control channel is given a notification type of ”RTCP”. The notification format below should make this a little simpler to comprehend.

Table 1. ElvinRTP notification format

ElvinRTP	1.0
channel	DW-TV
type	DATA
content-type	video
participant	myUniqueID
data	rtp packet

The content-type field specifies what kind of data is being sent in this RTP session. The participant field is globally unique within an RTP session. It may be generated by the ElvinRTP transport layer, or it may use the SSRC field from the RTP layer. This field is required so that packets originating from a participant are not routed back to that same participant. So a subscription will always be of the form “channel == channel X where participant != myUniqueID”. Finally, the data field contains the RTP packet itself.

Note that Elvin can use server discovery to locate the nearest Elvin router. This feature could be used to allow absolute location transparency from the client’s point of view. The client does not need to know the source of the multimedia content, and it isn’t required to know the address of an Elvin server.

5.5. Teletext Notifications

Unlike the video and audio components of the original DVB/MPEG-2 source, the teletext is not sent via RTP. Instead, it teletext is sent as a string in its own kind of Elvin notification. DVB teletext can be sent as character data or it can be bitmapped. This architecture works only with character-based teletext. The Content Server processes the

teletext received from the DVB broadcast and sends an Elvin notification, which contains the channel name and the teletext string.

5.6. The Client

RFC 2250 gives the name Receiving Interworking End-System to those nodes which receive MPEG data over RTP. In other words, they are basic clients which do not transmit any MPEG data themselves. These are the target nodes for our architecture.

As will be described in more detail in Section 6, the client establishes a connection to an ElvinRTP transport address. This is will be the name of a TV channel. However, depending on the user's needs, the user may not initially know the name of the channel to which they want to connect. They may just have an idea of the *content* they want to view. In this case, they subscribe to receive teletext notifications that match the content they wish to see. The client application then retrieves the channel name from the teletext notification and connects to that channel. The teletext notification may come via the correlator depending on the type of customisation involved. A correlation expression contains (i) an event section that describes what content the client is interested in. The event section describes the correlation between the teletext data from one channel and the teletext from another channel, or another Elvin based application and (ii) an action section that specifies how to re-emit the relevant teletext notifications. The client will organise for the notification to be emitted on a channel which it shares with the correlation engine.

6. Client Interface

This section provides the subscription expressions and correlation specifications needed to implement the motivating examples in Section 2.

6.1. The User Knows the Channel

This is the basic scenario. Here, the user explicitly connects to a specific channel with a well known name. All subsequent scenarios must ultimately connect to a TV channel in the same way once the client application has determined the name of the channel to connect to. As dictated by the RTP standard, each RTP session has a data channel and a control channel. Furthermore, the video and audio content should be sent in separate RTP sessions. Thus, we have two sessions, each with its own data and control channels, yielding four separate connections. Each of the subscription expressions below corresponds to one of the four required connections.

- require("ElvinRTP") && "channel"=="DW-TV" && "type"=="data" && "content-type"=="video" && "participant" != "12345".
- require("ElvinRTP") && "channel"=="DW-TV" && "type"=="rtcp" && "content-type"=="video" && "participant" != "12345".
- require("ElvinRTP") && "channel"=="DW-TV" && "type"=="data" && "content-type"=="audio" && "participant" != "12345".
- require("ElvinRTP") && "channel"=="DW-TV" && "type"=="rtcp" && "content-type"=="audio" && "participant" != "12345".

The last component of each subscription ensures that the Elvin router does not "echo" the client's own notifications back to the client.

6.2. Quick! Turn on Channel 9!

The final scenario provides just one example of correlating multimedia content with content from other applications. It demonstrates the way in which Elvin can be used to create scenarios which were not envisaged at the time the client application was developed. It also demonstrates the power of composing two different Elvin based applications to create a useful tool.

Imagine you have established a private Elvin based chat group with your friends. For example, you might be using the Elvin xtickertape program. You have agreed that the sequence of notifications "ALERT channel" followed by "<channel name>" is to be used to indicate there is something exciting on a certain channel. Whenever a friend sends a message like this to the chat group, you want your TV viewing application to automatically connect to the specified channel with no intervention by you.

In this case, we need to correlate notifications from two entirely separate applications: the MPEG streaming application and the chat program. The channel name sent in a chat notification needs to be correlated with an actual channel that transmits MPEG data. The following correlation specification will yield the desired effect:

```
<rule name=''news-alert''>
  <and guard=''events['mpeg']['channel'] ==
       events['chat2']['TICKERTEXT']''/>
    <event name=''mpeg'' subscription=
      ''require(ElvinRTP)'' />
    <then guard=''events['chat1']['USER'] ==
         events['chat2']['USER']''/>
      <event name=''chat1'' subscription=
        ''require(Chat) && TICKERTEXT ==
          'ALERT channel' && (USER == 'Bill'
           || USER == 'Anna')''/>
      <event name=''chat2'' subscription=
        ''require(Chat) && (USER == 'Bill'
          || USER == 'Anna')'' />
    </then>
  </and>
</rule>
```

This correlation expression will allow Bill and Anna to "turn on" a TV channel on your workstation. Note that no extra functionality is required to be programmed into the client. This scenario is made possible by Elvin and the Elvin correlation engine.

Of course there are certain security issues with this scenario (how do we know that someone isn't spoofing notifications that claim to be from Anna?) that we will ignore. However, Elvin contains security measures that will overcome these kinds of security threats. A discussion of these measures is beyond the scope of this paper.

7. Conclusion

This paper has presented a scheme for personalising multimedia feeds dynamically. It is this feature that distinguishes it from other similar architectures. We have shown that content-based event notification systems are a natural choice for implementing such an architecture. By using the Elvin notification protocol, we have designed a system whereby content from multiple TV channels can be filtered and correlated with each other, and with events from completely independent applications.

The authors' continuing interests lie in the application of this architecture to mobile environments and the inclusion of context-awareness. It is envisaged that multimedia feeds could be automatically personalised based upon the mobile user's current context. It is expected the choice of a loosely coupled, asynchronous protocol such as Elvin will be advantageous in mobile environments where disconnectedness is a major issue.

References

[1] C. D. Basso, R. Cranor, M. Gopalakrishnan, C. Green, D. Kalmanek, S. Shur, C. Sibal, Sreenan, and J.E. van der Merwe. Prism, an ip-based architecture for broadband access to tv and other streaming media. In *Proceedings of IEEE International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, 2000.

[2] Tim Bray, Jean Paoli, and C. M. Sperberg-McQueen (eds). Extensible markup language (XML) 1.0. W3C Recommendation, 1998.

[3] Martin G. Brown, Jonathon T. Foote, Gareth J. F. Jones, Karen Sparck Jones, and Steve J. Young. Automatic content-based retrieval of broadcast news. In *ACM Multimedia 95 Electronic Proceedings*, November 1995.

[4] A. Carzaniga, D. Rosenblum, and A. Wolf. Interfaces and algorithms for a wide-area event notification service, 1999.

[5] G. Cugola, E. Di Nitto, and A. Fuggetta. The JEDI Event-based Infrastructure and its Application to the Development of the OPSS WFMS. *IEEE Transactions on Software Engineering*, 2001.

[6] Michael Henderson. A framework for event correlation. Master's thesis, University of Queensland, http://elvin.dstc.edu.au/projects/correlation/index.html, October 1999.

[7] D. Hoffman, G. Fernando, V. Goyal, and M. Civanlar. RTP Payload Format for MPEG1/MPEG2 Video . IETF Internet Standard, January 1998.

[8] M. Wray and R. Hawkes. Distributed virtual environments and VRML: an event-based architecture. In *Proceedings of the Seventh International WWW Conference (WWW7)*, 1998.

[9] S. Maffeis. iBus – The Java Message Bus. Available at http://www.softwired.ch.

[10] Masoud Mansouri-Samani and Morris Sloman. Gem: A generalised event monitoring language for distributed systems. *IEEE/IOP/BCS Distributed Systems Engineering Journal*, 4(2):96–108, June 1997.

[11] Mark Hapner and Rich Burridge and Rahul Sharma and Joseph Fialli. Java Message Service. Technical report, Sun Microsystems, Inc, http://java.sun.com/products/jms/docs.html, 2001.

[12] OMG. Notification Service Specification. Technical report, Object Management Group, ftp://ftp.omg.org/pub/docs/formal/00-06-20.pdf, 2000.

[13] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications . IETF Internet Standard, January 1996.

[14] B. Segall and D. Arnold. Elvin has Left the Building: a Publish/subscribe Notification Service with Quenching . In *Proceedings of AUUG97*, Brisbane, Australia, September 1997.

[15] ETSI standards. Specification for conveying itu-r system b teletext in dvb bitstreams. ETSI Blue Book References A041, 1999.

[16] ETSI standards. Implementation guidelines for the use of mpeg-2 systems, video and audio in satellite cable and terrestrial broadcasting applications. ETSI Blue Book References A001 - TR 101 154, 2000.

[17] G. Banavar T.D Chandrea B.Mukherjee J. Nagarajarao R.E Strom D.C Sturman. An efficient multicast protocol for content-based publish-subscribe systems. In *The 19th IEEE International Conference on Distributed Computing Systems (ICDCS'99)*, Austin, TX YSA, May 1999.

[18] Stuart Wray, Tim Glauert, and Andy Hopper. The medusa applications environment. In *International Conference on Multimedia Computing and Systems*, 1994.