Abstract Service Definition for Media Services

- Learn about abstract service interfaces for digital media
- Implement an ASD compliant Web service
- Integrate digital media in SOA enabled business processes

Philip Monson
Paolo Dettori
Peter Guglielmino
Chung T. Nguyen
Tam N.C. Nguyen
Julio Nogima
Frank A. Schaffa

Redpaper

ibm.com/redbooks

International Technical Support Organization

Abstract Service Definition for Media Services

April 2009

REDP-4464-00

Note: Before using this information and the product it supports, read the information in “Notices” on page v.

First Edition (April 2009)

This edition applies to WebSphere Enterprise Service Bus with Media Extender v6.2, WebSphere Process Server v6.2, WebSphere Service Registry and Repository v6.2, WebSphere Business Modeler v6.2, WebSphere Business Monitor v6.2, WebSphere Integration Developer v6.2, Tivoli Composite Application Manager for SOA v7.1.

Contents

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at http://www.ibm.com/legal/copytrade.shtml

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

IBM®	Notes®	Redpaper™
Lotus Notes®	Redbooks®	WebSphere®
Lotus®	Redbooks (logo) ®	

The following terms are trademarks of other companies:

eXchange, Java, JRE, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, SQL Server, Visual Studio, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Preface

This IBM® Redpaper™ publication presents the Abstract Service Definition (ASD) for media, which realizes the core proposition of services-oriented architecture (SOA) by providing a flexible and extensible Web service mechanism to access classes of media services, independently from the particular implementation of each service at an appropriate level of abstraction.

This work extends the value proposition that IBM is bringing forward with Media Hub Solution Framework, which provides the foundation for a business-led approach to reducing costs and getting new products and services to consumers more rapidly through improved production and distribution capabilities.

This document is intended for business and IT professionals who are responsible for architecting and implementing the next generation of production and distribution systems.

The team that wrote this paper

This paper was produced by a team of experts from around the world working at the IBM T.J. Watson Research Center in Hawthorne, NY.

Philip Monson is a Business Development Project Leader for the IBM ITSO. Phil has been with Lotus® and IBM for 18 years, joining the company when the early versions of Lotus Notes® were rolled out for internal use. He has served in management, technical, and consulting roles in the IT, Sales, and Development organizations.

Paolo Dettori is a Senior Software Engineer at IBM Research with more than 16 years of experience in the area of multimedia technologies. He received his M.S. in 1991 in electrical engineering from the Polytechnic University of Turin, Italy, and joined IBM in 1992. Since then, he has worked as Lead Architect on a number of projects focused on middleware technologies for multimedia applications. Currently, he is a Lead Architect for the Media Hub project at IBM Research. Mr. Dettori authored and co-authored several journal and conference papers and has five patents issued in the U.S. on multimedia technologies.

Peter Guglielmino is the CTO of the IBM Media and Entertainment Industry. Peter is responsible for developing the architectures that serve as the basis for the global digital media offerings relating to digital media archives, rich media utilities, media on demand, and secure content distribution networks. Peter has designed the digital archive solutions for Coca-Cola, Imagebank, Pearson Broadband, and Ogilvy & Mather's digital video archive. Currently, Peter is working with IBM Research to extend the Enterprise Services Bus properties of persistence, mediation, transformation, and routing to media by incorporating the MPEG-21 standard providing a media-aware SOA infrastructure. Peter also has provided assistance regarding digital audio and video technologies throughout IBM.

Chung T. Nguyen is a Senior WebSphere® Integration Solution Architect for the WebSphere Business Development organization. He has been working with many IBM Business Partners over the last 8 years. In addition, he spent several years working in the IBM Java™ Technology Center and Pervasive Computing. He has a Masters degree in Electrical Engineering. He also has an MBA. One of his main interests is to create innovative technology solutions from product integrations between IBM and its Business Partners.

Tam N.C. Nguyen is a Senior Managing Consultant at IBM with a Master I/T Architect Certification from The Open Group and has more than 17 years of IT experience. He joined IBM in 1993. His technical areas of expertise include digital media, enterprise IT transformation, Enterprise Application Integration (EAI), and service-oriented architecture (SOA). He has authored and co-authored several IBM internal published papers and presented at conferences focusing on enabling digital media, identity management, and SOA solutions for the enterprise. Currently, he is part of the IBM Software Services Organization, working with IBM customers to leverage IBM software in their architecture or IT strategy.

Julio Nogima is a Software Architect at the IBM T.J. Watson Research Center in Hawthorne, NY. He has a degree in Electrical Engineering and his research interests include SOA-based systems and middleware for processing and distribution of multimedia content. His current work is focused on improving the existing SOA middleware capabilities to provide better support for digital media applications and services.

Frank A. Schaffa is a Research Staff Member at IBM Research. He received Electrical Engineering and MSc degrees from Escola Politécnica University of São Paulo, Brazil. He also received a Ph.D. in Computer Science from the University of California, at Los Angeles. He joined IBM T. J. Watson Research Center in 1990, where his current research activities include broadband networks, multimedia digital distribution and middleware media processing. He currently leads the team that is developing the Media Hub technology and related tooling. He has several publications and patents in the areas mentioned.

Thanks to the following people for their contributions to this project:

Tim Banks, Software Architect, Media Extender for WebSphere ESB, IBM Software Group

Martin Keen, Senior IT Specialist, WebSphere, IBM ITSO

Subramanian V. Ganesh, IT Architect, Media Entertainment, IBM Software Group

Ann Reiten, Vice President, Global Media & Entertainment Solutions IBM

Dieter A. Haas, Media &Telco IT Architect, IBM

Edward Bottini, Global Partner Leader, Industry Solutions and Retail, IBM Software Group

Norman Korn, Industry Solutions Enablement Manager, IBM Software Group

Wangming Ye, Senior Software Engineer, Business Partner Technical Strategy and Enablement, IBM Software Group

Become a published author

Join us for a two- to six-week residency program! Help write a book dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You will have the opportunity to team with IBM technical professionals, Business Partners, and Clients.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you will develop a network of contacts in IBM development labs, and increase your productivity and marketability.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our papers to be as helpful as possible. Send us your comments about this paper or other IBM Redbooks® in one of the following ways:

- Use the online **Contact us** review Redbooks form found at:

  ibm.com/redbooks

- Send your comments in an e-mail to:

  redbooks@us.ibm.com

- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400

1

Abstract Service Definition overview

In this chapter, we introduce the concept of an Abstract Service Definition (ASD). The ASD is an architected service interface, specifically designed for ease of use. ASD defines categories of services, for example transcoders, data movers, watermarkers, repositories, and media analytics. The basic concept is that an ASD interface should be focused on function and provide a common terminology (ontology) to enable ease-of-use and reduce the system integration skill required to implement a media-enabled service-oriented architecture (SOA) deployment.

1.1 ASD business value

A core value proposition of SOA is to offer software developers and system integrators a simple way to assemble composite applications from a set of reusable components. This approach reduces integration and maintenance costs, and allows the business to define how composite applications should be assembled to best meet the challenges it faces.

Moving to standards-based interfaces such as Web services is a major step forward in realizing the goal of loosely coupled heterogeneous distributed applications. However, in many instances, especially in the case of existing applications that implement a Web service interface, this is not sufficiently coarse-grained and offers little advantage over the earlier API it was meant to augment.

In this paper, we present the Abstract Service Definition (ASD) for media. The ASD provides a flexible and extensible mechanism to access classes of media services independently from the particular implementation of each service at an appropriate level of abstraction. This technique allows the realization of the SOA principles, simplifies the orchestration and provides the appropriate mediation, taking into consideration the complexity of media operations and associated metadata.

In developing these standards-based interfaces, we identify three major stakeholders: the independent software vendor (ISV) who is providing the desired function, the customer who purchases these functions based on customer requirements, and the integrator that *packages* the overall solution. The customer can be further divided into the technical and the business stakeholders.

The benefits that are common to all stakeholders are:

- The basic value of a componentized SOA to provide the simple assembly of reusable components into composite applications
- The use of standard interfaces to simplify the integration of the components into the loosely coupled composite applications that meet the business requirements
- The value of reusable components that have been reviewed and tested by many developers
- The value in eliminating *unusual* or *non-conforming APIs* by wrapping them to a *new* standardized interface based on business requirements

The benefits that an ISV receives from implementing an ASD instead of developing a proprietary interface are:

- The interface-to-function definition is based on industry standards, which have been developed in collaboration with multiple customers and ISVs, releasing the individual ISV from the development effort of creating the ISV's own interface.
- The ISV can spend development effort on the function, not on the interface. The quality of the function that the service provides is the differentiator that customers are interested in, not how to communicate with the service.
- Using an ASD eases the selling of the service because of less integration effort and a lower learning curve to include the service into a composite application.
- The ASD provides common errors or faults as part of the interface, again requiring lower development cost.
- The use of an ASD makes it easier to provide an acceptance test, performance, and function package.

- Wrapping a non-conforming API and converting it into a conforming API is usually a simple process.
- The ASD provides the ability to add extensions to the API for further functionality enhancements

The technical customer benefits from the use of ASDs in the following ways:

- A standard interface allows for replacement and upgrade independence.
- The ASD decouples the functional interface from a particular provider implementation of the function.
- Abstracting the service interface makes it easier to scale up and diversify.
- The ASD leverages the concept of a registry which allows for single point of management of schemes, profiles, and metadata.
- The ASD enables defining generic workflows templates, which can be used as media process patterns.
- The ASD standardized interface makes governance and monitoring (such as auditing and common fault values) easily understood for fast reaction and possible automated fault recovery.

The business customer benefits are:

- The ASD provides better governance and monitoring of resources such as utilization, faults, and quality, allowing for better predictable workflows.
- The ASD leverages industry technology, providing a greater pool of expertise that might be available in the customer development shop.
- Return on invest (ROI) advantages are possible. ROI advantages can be simpler and lower cost integration, and easier performance and cost analysis.

Integrators can realize higher productivity and reduce cost through the use of ASDs because:

- Use of a standard interface eliminates having to learn different interfaces for multiple ISVs that deliver the same service (such as transcode, transport, and watermark).
- The ASD simplifies integration because the interface is well known and adheres to standards.
- After an ASD interface is mastered, it allows for faster reuse of development activities across multiple engagements.
- The ASD provides a best practice for the definition of a service definition

1.2 Scope

The ASD covers only the interface design. It does not cover the specific details of the service implementation, such as the need for transactionality in some interactions (repository operations, for example), which are not covered in this document. Furthermore, this document does not address the general aspects of design and development methodology for service connectors and how they relate to established practices as service-oriented analysis and design (SOMA or Service-Oriented Modeling and Architecture). In addition, this document does not address the life cycle of an ASD.

From a technical perspective, the scope of the ASD API is restricted to a network domain where callbacks from the Web service to the Web service client are permitted. As a result, If the client is protected by a firewall, inbound callback operations can be blocked, unless the firewalls involved are properly configured.

1.3 ASD concepts

Figure 1-1 illustrates the high-level architecture for an ASD Web service. A service provider (such as a transcoder) offers a service provider API that is specific to the implementation of the service. A higher-level ASD Web service interface provides an abstraction layer that allows business processes and Web service clients to invoke the provider without prior knowledge of specific implementation details. This approach promotes reuse and the ability to swap and select service providers without any effect on the dependent business processes.

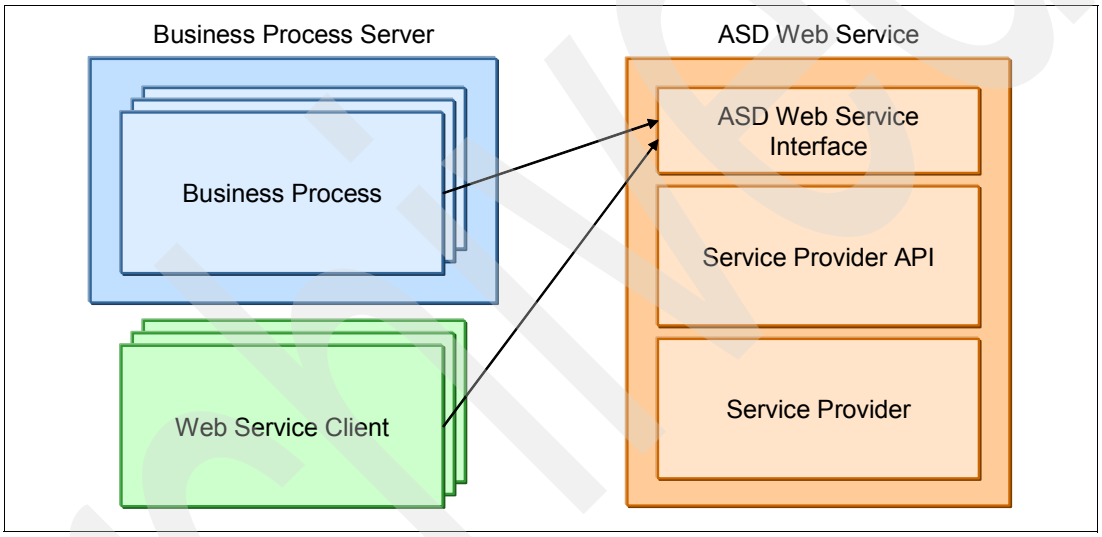

Figure 1-1 High-level architecture

1.3.1 General features of the ASD

The ASD provides a flexible and extensible mechanism to access classes of media services independently from the particular implementation of each service. Each service class provides a different interface, however, a number of common features are shared by all services classes:

Operations

Media-related operations involve the manipulation of large media files. Such files might take a long time to transfer across services and to process. As a result, most media-related operations are long-running and have to be asynchronous. A key aspect of the ASD API is the support of asynchronous operation for the service binding as standardized by the Web Services Interoperability Organization (WS-I) in Basic Profile, Version 1.0 and Version 2.0 (Working Draft).[1] The WS-I Basic Profile provides a set of Web Services specifications aimed to promote interoperability across different vendors. The WS-I standard mandates the use of the hypertext transport protocol (HTTP) with SOAP binding.

[1] The standard is available at http://www.ws-i.org/deliverables/workinggroup.aspx?wg=basicprofile

The support of long-running, asynchronous operations with SOAP/HTTP binding can be implemented by two different interaction patterns:

- An implicit asynchronous pattern
- A synchronous request/response with asynchronous notification pattern

Implicit asynchronous pattern (WS-Addressing)

The implicit asynchronous pattern is based on the The W3C Web Services Addressing (WS-Addressing) standard. The W3C WS-Addressing standard, *Web Services Addressing 1.0 - Core W3C Recommendation* provides asynchronous support with SOAP/HTTP binding. It is available from:

http://www.w3.org/TR/ws-addr-core/

An asynchronous call with SOAP/HTTP binding and WS-Addressing results in a call to the Web service that implements the ASD and an asynchronous callback from the service to the invoking client to return the result of a specific media operation. Figure 1-2 illustrates a typical scenario for the asynchronous invocation of an ASD service with WS-Addressing. In this scenario, a Web service client has to implement a client stub, used for invoking the service according to the WS-Addressing specifications, and a server stub, used to receive the asynchronous reply back from the service. The service can then take advantage of available WS-Addressing runtimes to support the asynchronous operation. Examples of Web service runtimes supporting WS-Addressing are the IBM WebSphere Application Server 6.x Web Services runtime and Axis 2 for Apache Tomcat.

During the initial synchronous interaction, the client sends to the service an endpoint used by the service for the asynchronous response in the WS-Addressing header (for example, *replyTo*); the service stores this information and uses it later for the callback. Furthermore, the WS-Addressing header provides the *MessageID* field to correlate the asynchronous reply with the operation initiated by the request. The asynchronous reply must contain the value of this parameter in the *RelatesTo* field.

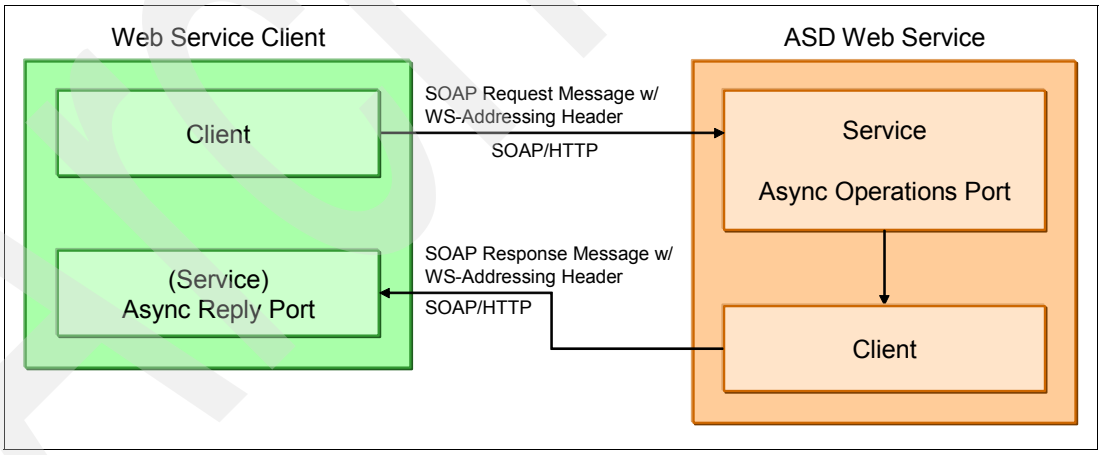

Figure 1-2 Asynchronous invocation of a ASD Service with WS-Addressing

Synchronous request/response with asynchronous notification pattern

This pattern is composed by a synchronous request/response call to the Web service implementing the ASD and an asynchronous call back from the service to the invoking client to return the result of a specific media operation. Figure 1-2 illustrates a typical scenario for the asynchronous invocation of an ASD service. In this scenario, a Web service client has to implement a client stub, used for invoking the service according to a traditional synchronous request-response pattern, and a server stub, used to receive the asynchronous reply back

from the service. Similarly, the service has to implement a server stub, to receive the synchronous request and to respond that the operation request has been received and processing is starting; and implement a client stub, used to call back the client when the operation is completed. The ASD implements this pattern by providing a port for the synchronous exchange, which should be implemented by the service, and a port for the asynchronous reply, which should be implemented by the client, as shown in Figure 1-3.

During the initial synchronous interaction, the client sends to the service an endpoint used by the service for the asynchronous response (*replyTo*); the service stores this information and uses it later for the call back. A set of correlation parameters (IDs) is adopted to correlate the asynchronous reply with the operation initiated by the initial synchronous exchange. These parameters must be present in the initial synchronous exchange (in the request, in the response, or in both) and in the asynchronous reply. The ASD API identifies a specific parameter used for correlation in the synchronous interaction and the asynchronous reply.

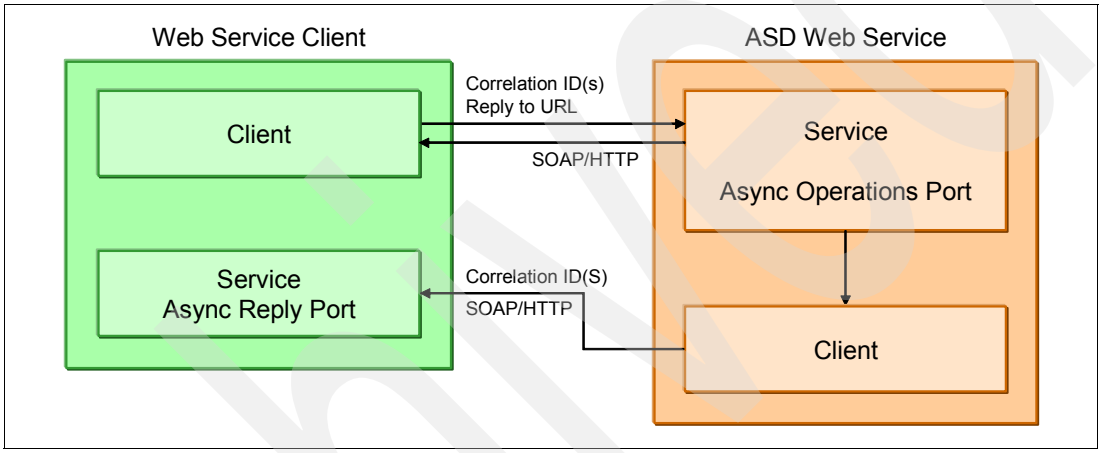

Figure 1-3 Asynchronous invocation of a ASD Service for the request/response/notification pattern

Status

An essential aspect for the management of long-running media operations is the ability to check the status of operations and interact with active queued jobs. For example, the following tasks might have to be performed for transcoding operations:

- Check the status of a job
- Cancel an active job
- Pause an active job
- Stop an active job
- Change priority of a job
- List jobs in a queue
- Clear a queue
- Delete single job from a queue
- Lock/unlock a queue

The adoption of a standard ASD interface for the status operations enable the use of common front end tools for media services management, and the ability to interact with running tasks from a client or workflow.

Content characteristics

Media services expose their functionality on a network using the standardized SOAP protocol.[2] The Media ASD extends the concept of SOAP message for media content by adding a new data type that represents a media asset. The new data type is based on the MPEG-21 standard, and in particular on the Digital Item Declaration (DID).[3]

Service metadata

The same ASD service of a class can be implemented by different vendors, resulting in different implementation instances for the same ASD service. Some implementation might require additional data, which is different from vendor to vendor. To accommodate vendor-specific metadata, the ASD API provides a Service Metadata element. This element, implemented by the XML schema `any` type, acts as a container for vendor-specific metadata, which is then provided as an XML file with its own XML schema. A key scenario for the adoption of service metadata is the one where a service is invoked through an intermediary sitting between the client and the Web service. The function of the intermediary is to lookup and select an appropriate service based on properties on the SOAP message sent from the client and properties of the available services belonging to a service class. The client is then presented with an abstract view of the invoked service and does not know the implementation details of each service. The intermediary is responsible to select the specific implementation and then insert the appropriate service metadata. An example of intermediary is a Web service mediation flow running on the IBM WebSphere Enterprise Service Bus (ESB). IBM provides specific mediation primitives to handle media services with the IBM Media Extender for ESB.

An example of service metadata for a transcoder service is illustrated in Example 1-1. In this example the metadata provides three properties that are specific to the transcoder implementation:

transcoderJobProfileName	Profile used by the transcoder for this job request
outputDirectory	Directory where transcoded files should be placed
outputBaseFilename	Base name for the transcoded files produced in output

Example 1-1 Service metadata sample for a transcoder service

```
<?xml version="1.0" encoding="UTF-8"?>
<s1:TranscoderMetaData xmlns:s1="http://metadata.transcoder.asd.ibm.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://metadata.transcoder.asd.ibm.com TranscoderMetadata.xsd ">
  <requester>requester</requester>
  <MetaData>
    <transcoderJobProfileName>transcoderJobProfileName</transcoderJobProfileName>
    <outputDirectory>outputDirectory</outputDirectory>
    <outputBaseFilename>outputBaseFilename</outputBaseFilename>
  </MetaData>
</s1:TranscoderMetaData>
```

[2] SOAP 1.1, W3C Standard, available at http://www.w3.org/TR/soap/
[3] MPEG-21 Overview, available at http://www.chiariglione.org/mpeg/standards/mpeg-21/mpeg-21.htm

1.3.2 MPEG-21

The ASD adopts the ISO MPEG-21 standard for representing media content. The MPEG-21 DID provides an abstract and lightweight model for declaring digital items, and an XML schema to represent the model in XML. In comparison to other industry standards such as the Material eXchange™ Format (MXF),[4] the DID offers the advantage of placing metadata separately from the content essence. A set of logical entities provide the basic building blocks for defining the digital item. Media files are represented with resources; a resource is referenced by a URI that identifies the asset and provides network access.

DIDL structure

The following list describes the key building blocks of the MPEG-21 DID standard:

Resource A resource represents a digital asset such as a video file, image, audio clip, or a textual asset. More in general, a resource might be used to represent a physical asset with a unique ID.

Component A component is the grouping of one or more equivalent resources. A component can then be associated to a set of descriptors. These descriptors provide metadata information related to all the resource entities grouped in the component.

Item An item is the grouping of one or more items, components, or both. An item can then be associated to a set of descriptors. These descriptors contain metadata information about the represented item. According to the standard, items are the first point of entry to the content for a user.

Container A container is the grouping of one or more items, containers, or both. A container can then be associated to a set of descriptors. The containers can be used to form groupings of Items. The descriptor metadata contain information about the represented container.

Descriptor A descriptor, in conjunction with a statement construct, typically associates textual metadata information with the object containing the descriptor itself. For example, a descriptor inside a component provides metadata about the component and the resources grouped by the component. Example of metadata might include textual information about the digital items (for example, title, date, author, format information) or licensing and digital rights.

For a more detailed description, see *MPEG-21 Overview*:

http://www.chiariglione.org/mpeg/standards/mpeg-21/mpeg-21.htm

Figure 1-4 on page 9 provides a Unified Modelling Language (UML) diagram of the key elements of the MPEG-21 DID standard.

[4] *MXF - The Material eXchange Format,* Bruce Devlin, EBU TECHNICAL REVIEW, July 2002, available at http://www.ebu.ch/en/technical/trev/trev_291-devlin.pdf

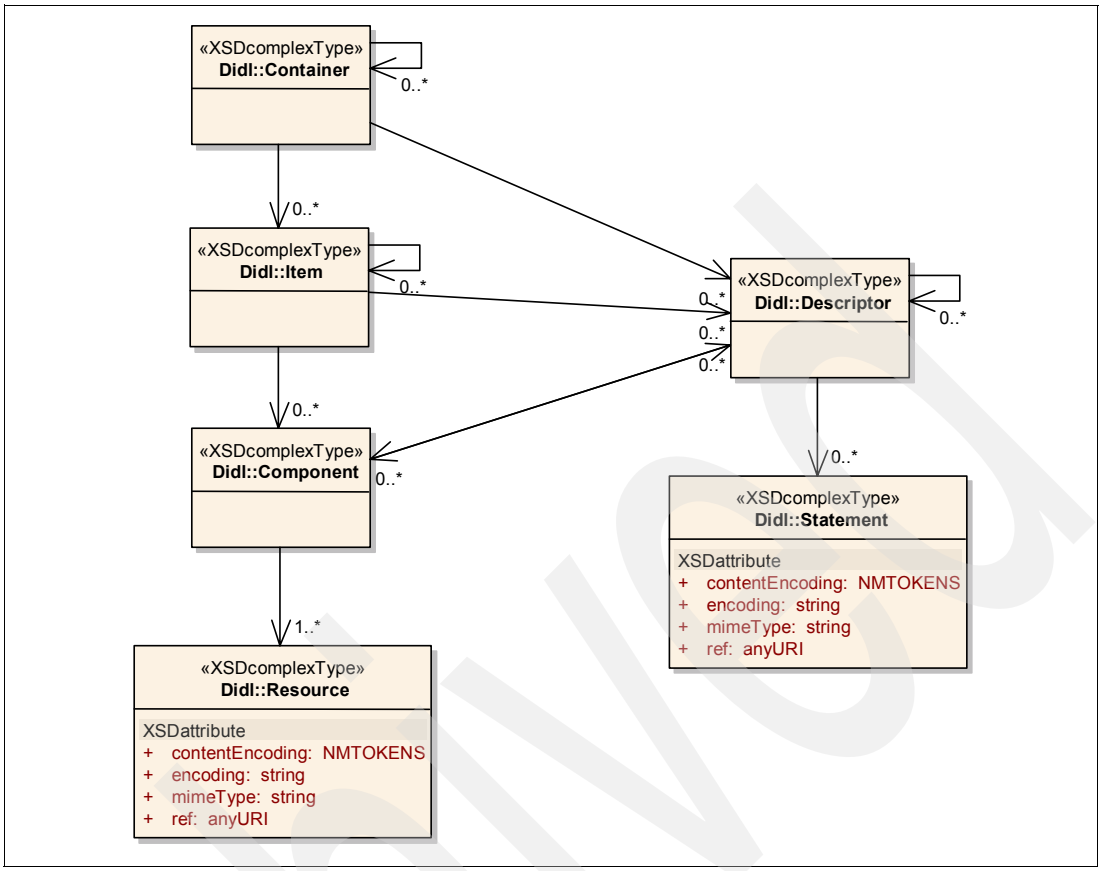

Figure 1-4 MPEG-21 DID Structure

A simple example of an MPEG-21 file is shown in Example 1-2, where all key elements are illustrated. Note how the `Resource` element holds the reference to the media file in the `ref` attribute. Note also how the `Descriptor/Statement` construct provides a placeholder where XML metadata for the media content can be placed.

Example 1-2 MPEG-21 DID sample

```
<?xml version="1.0" encoding="UTF-8"?>
<didl:DIDL xmlns:didl="urn:mpeg:mpeg21:2002:02-DIDL-NS"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="urn:mpeg:mpeg21:2002:02-DIDL-NS didl.xsd ">
 <didl:Container>
  <didl:Item>
   <didl:Component>
    <didl:Descriptor>
     <didl:Statement mimeType="text/xml; charset=UTF-8">
      <!-- Metadata goes here -->
     </didl:Statement>
    </didl:Descriptor>
    <didl:Resource mimeType="video/quicktime"
     ref="ftp://ftpserver.ibm.com/sample.mov" />
   </didl:Component>
  </didl:Item>
 </didl:Container>
</didl:DIDL>
```

Media Descriptor

One key benefit of adopting Web service and implementing SOA concepts is the ability to make runtime decisions for selecting the most suitable service instance of a service category. For example, you might want to select the transcoder service most suitable for a particular video file. In this case, the information about the mime type provided by the MPEG-21 `Resource` element might not be enough. This consideration provides the rationale for a media descriptor providing additional information about the format of the media file referenced by the `Resource` element. The media descriptor provides information that is not directly inferred from the mime type or the file extension and that is useful for identifying a particular media format. Although very few standards exist that provide information about the encoding properties of media files, one of them, MPEG-7, provides certain basic properties in the Media Information Descriptor such as file size, frame size, and coding. However, it does not cover characteristics such as encoding mode, if progressive, codec information, codec version, codec, family and so on, which are relevant in characterizing one particular media format. To cover these additional media properties, we have defined a media descriptor, which is described in this section. The use of the media descriptor specified here is recommended for ensuring that the ASD services are compatible with the Media Hub solution and for promoting an ecosystem where formats are referenced without ambiguities.

An example of a media descriptor for a QuickTime file is provided in Example 1-3.

Example 1-3 Media Descriptor sample with most common parameters

```
<?xml version="1.0" encoding="UTF-8"?>
<md:MediaDescriptor xmlns:md="http://md.wemx.ibm.com"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://md.wemx.ibm.com md.xsd ">
 <MediaFormat ID="qtmov-mp4-hbr" extension="mov" family="MPEG-4"
format="QuickTime"
  info="QuickTime movie">
  <VideoFormat>
   <Frame aspectRatio="1.500" height="480" rate="29.970" width="720" />
   <Encoding mode="CBR" progressive="false" />
   <Codec name="2Vuy" />
   <Bitrate>79000000</Bitrate>
  </VideoFormat>
  <AudioFormat>
   <Codec name="PCM" />
   <Bitrate>384000</Bitrate>
   <Channels>2 channels</Channels>
   <Resolution>16</Resolution>
  </AudioFormat>
 </MediaFormat>
</md:MediaDescriptor>
```

The most important attributes for `MediaFormat` are the ID, the format and extension attributes. They identify the wrapper (such as QuickTime), which contains the media components (such as video and audio streams). The other relevant elements are the `VideoFormat` and `AudioFormat` tags. The most important features of each element (for format compatibility checking) are the name attribute of the `Codec` element. The other elements provide specific format instance information (such as resolution, bit rate, encoding mode, and others.). The information contained in the `MediaFormat` element can be obtained by using tools that analyze the physical media file (such as MediaInfo). For more information about MediaInfo, see:

http://sourceforge.net/projects/mediainfo

One important aspect of the Media Descriptor is the ability to assign an ID to a particular media format. This approach allows you to maintain the description of a format in a separate registry and provide just the ID as a key to a particular format. The simplest Media Descriptor should contain the `MediaFormat` with at least the `ID` and the `extension` attributes. This case is shown in Example 1-4. Note that if an ID is assigned to each format, it is not required to carry all the other information required by the MediaFormat elements. In this case, the MediaFormat ID can be used as a reference to a Media Descriptor (with all related media information) maintained in a central repository.

Example 1-4 Media Descriptor sample with ID only

```
<?xml version="1.0" encoding="UTF-8"?>
<md:MediaDescriptor xmlns:md="http://md.wemx.ibm.com"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://md.wemx.ibm.com md.xsd ">
 <MediaFormat ID="qtmov-mp4-hbr" extension="mov"/>
</md:MediaDescriptor>
```

A Media Descriptor should be placed in a `Descriptor/Statement` inside the MPEG-21 `Component` object containing the referenced resources.

ResultMapList

When a service performs an operation on a digital media file, it can produce a new media content derived from the source content. For example, a transcoder might receive a media file in MPEG-2 format and produce one or more new media file(s) in MPEG-4 format. An ASD service should not provide in output the source content used for a particular operation, but only the output content. For auditing purpose, being able to reconstruct the relationship between the input and the output content can be useful. This information is not required by an ASD adapter for the purpose of processing the media content, but can be used by auditing and logging tools for tracking the life cycle of media content across multiple service invocations. The `ResultMapList` provides the relationship between the reference to the content in input to and in output from an operation. A compliant ASD service is expected to update the `ResultMapList` to indicate what output is created from a particular input, but it should not use the information in the `ResultMapList` for any processing decision.

A simple example of a `ResultMapList` is illustrated in Example 1-5. In this example, the file referenced by the URL `file://server1/c:/movieIn1.mpg` is the input to a transcoding operation which produces the two files referenced by the URLs:

- `file://server2/c:/movieOut11.mpg`
- `file://server2/c:/movieOut12.mov`

Example 1-5 ResultMapList sample

```
<?xml version="1.0" encoding="UTF-8"?>
<rml:ResultMapList xmlns:rml="http://rml.wemx.ibm.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://rml.wemx.ibm.com rml.xsd ">
  <MapOneToMany>
    <source>file://server1/c:/movieIn1.mpg</source>
    <target>file://server2/c:/movieOut11.mpg</target>
    <target>file://server2/c:/movieOut12.mov</target>
  </MapOneToMany>
</rml:ResultMapList>
```

Because certain media operations can take one file at the input and produce multiple files at the output, and some other operations can take multiple files in input and produce only one output, the schema provides two complex types (`MapOneToMany` and `MapManyToOne`) to represent these two cases.

It is worth noticing that the reference to each resource is pointing to the URL provided in the `ref` attribute of the MPEG-21 resource element. Because the `source` element is referencing a resource that is no longer in the MPEG-21 container, an XPath expression to that resource in the MPEG-21 XML cannot be used. Therefore, use the URL (ref) to each resource to identify it in the ResultMapList.

A `ResultMapList` should be placed in a `Descriptor/Statement` inside the MPEG-21 Container object containing the referenced digital items.

Putting it all together

The MPEG-21 standard offers a great deal of flexibility for describing complex media structures. Because the standard is extremely open, there might be issues of compatibility between ASD services if some basic guidelines is not enforced. ASD services should follow the guideline schematically illustrated in Figure 1-5 on page 13. The figure also provides a complete example of MPEG-21 DID object, which includes the Media Descriptor and the `ResultMapList` for two items. The general guidelines for ASD recommend the following:

- A DID should have no more then one `Container` element.
- The `Descriptor/Statement` element with the `ResultMapList` should be placed inside the `Container` element.
- The `Container` can have multiple `Item` elements. Each `Item` element represents only one physical resource, and contains one `Component` element with one `Resource` element.
- The `Descriptor/Statement` element with the `MediaDescriptor` should be placed inside the `Component` element.

Note that the interoperability between ASD services following these guidelines and other MPEG-21 descriptions can be obtained by defining XSLT transformations which convert a generic MPEG-21 structure in an ASD-compliant structure. Such transformations also ensure compatibility with the IBM Media Extender for ESB, which currently operates on MPEG-21 objects following the above guidelines with the further restriction of a single `Item` for each `Container`.

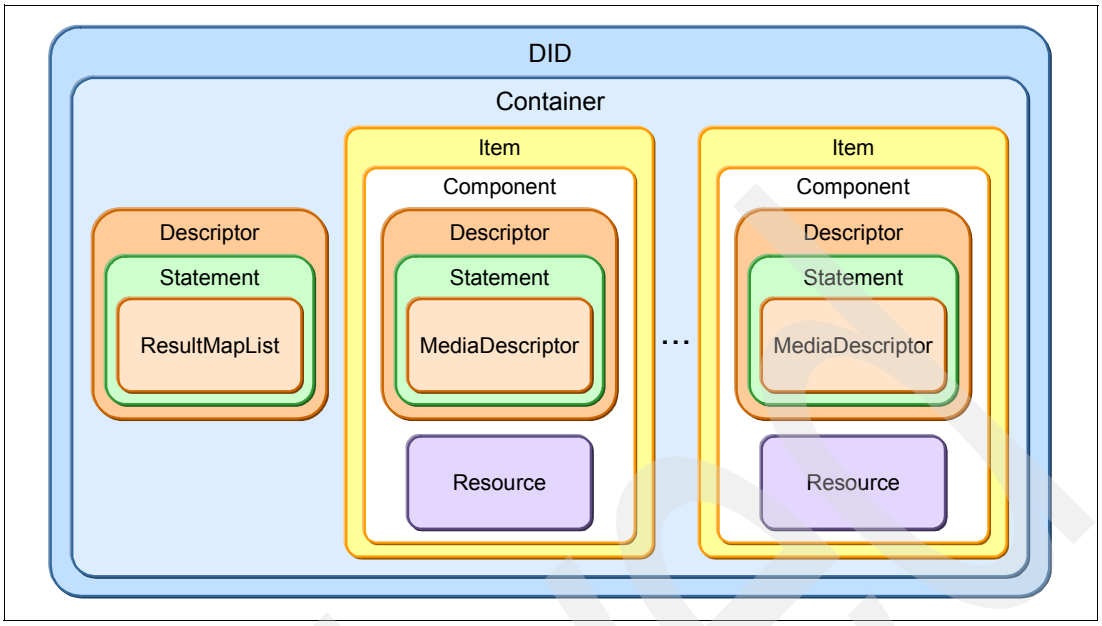

Figure 1-5 MPEG-21 DID with Media Descriptor and ResultMapList

The complete DID XML for this example is shown in Example 1-6.

Example 1-6 Complete DID XML example

```
<?xml version="1.0" encoding="UTF-8"?>
<didl:DIDL xmlns:didl="urn:mpeg:mpeg21:2002:02-DIDL-NS"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="urn:mpeg:mpeg21:2002:02-DIDL-NS didl.xsd ">
 <didl:Container xmlns:didl="urn:mpeg:mpeg21:2002:02-DIDL-NS">
  <didl:Descriptor>
   <didl:Statement mimeType="text/xml; charset=UTF-8">
    <rml:ResultMapList xmlns:rml="http://rml.wemx.ibm.com"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://rml.wemx.ibm.com rml.xsd ">
     <MapOneToMany>
      <source>file://server1/c:/movieIn1.mpg</source>
      <target>file://server2/c:/movieOut11.mpg</target>
      <target>file://server2/c:/movieOut12.mov</target>
     </MapOneToMany>
    </rml:ResultMapList>
    </ResultMapList>
   </didl:Statement>
  </didl:Descriptor>
  <didl:Item>
   <didl:Component>
    <didl:Descriptor>
     <didl:Statement mimeType="text/xml; charset=UTF-8">
      <md:MediaDescriptor xmlns:md="http://md.wemx.ibm.com"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://md.wemx.ibm.com md.xsd ">
       <MediaFormat ID="mpg2-hbr"extension="mpg"/>
      </md:MediaDescriptor>
     </didl:Statement>
    </didl:Descriptor>
```

```
    <didl:Resource mimeType="video/mpeg" ref="file://server2/c:/movieOut11.mpg" />
   </didl:Component>
  </didl:Item>
  <didl:Item>
   <didl:Component>
    <didl:Descriptor>
     <didl:Statement mimeType="text/xml; charset=UTF-8">
      <md:MediaDescriptor xmlns:md="http://md.wemx.ibm.com"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://md.wemx.ibm.com md.xsd ">
       <MediaFormat ID="mov-lbr" extension="mov"/>
      </md:MediaDescriptor>
     </didl:Statement>
    </didl:Descriptor>
    <didl:Resource mimeType="video/quicktime"
ref="file://server2/c:/movieOut12.mov" />
   </didl:Component>
  </didl:Item>
 </didl:Container>
</didl:DIDL>
```

Note that Example 1-6 on page 13 is only a sample; much more complex MPEG-21 representations can be created, as long as they comply to the guidelines described previously.

Other standards

Relevant open standards in the area of content representation and packaging are:

- MPEG-21
- Material eXchange Format (MXF)
- Metadata Encoding and Transmission Standard (METS)
- Broadcast Metadata Exchange Format (BMF)

MXF is the standard with the most industry adoption, but it only provides a mechanism to transport metadata in a bitstream containing the metadata and the essence, making it impractical for use in the SOAP messages, because this approach would require sending the whole media file plus the metadata in each SOAP message. Both MPEG-21 and METS provide the ability to reference content and provide XML metadata in a separate container. METS is widely adopted in the digital library community, but with extremely limited adoption outside of the community. BMF is another emerging effort of standardization for the digital objects and metadata exchanged from the production process to the play-out and archiving phase. BMF focuses specifically on the Broadcasting environment and it is still in an early phase of adoption in several European companies. On the other hand, MPEG-21 DID has a broader scope, it has been endorsed by the MPEG and ISO organization, and a growing number of companies are starting to adopt it. Because MPEG-21 is an extremely lightweight and extensible standard for packaging and referencing content and metadata, it is quite straightforward to adopt it in conjunction with MXF, for example to reference MXF resources and provide a minimal set of metadata about them. In other words, those organizations that have already adopted MXF can benefit from the introduction of MPEG-21 standard used in conjunction with MXF to reference their media resources and metadata. For example, a MXF file can be the physical media file, described by MPEG-21. Furthermore, because MPEG-21 does not mandate a specific standard for the metadata format, but provides a standard mechanism to package content and metadata, it can be used to transport any XML metadata representation of MXF. A similar approach can be adopted for interoperability with the BMF metadata.

Future capabilities

MPEG-21 standardizes the manner in which resources and metadata associated to a digital object are represented (MPEG-21 DID). MPEG-21 also proposes a way to process this object by standardizing how to represent one or more operations on the object (MPEG-21 DIP[5]). An example of an operation might be to download a media object. Another example might be transcoding a media object to a more appropriate format for the user, and then downloading the transcoded media object. In both examples, the operation at the object level could be defined simply as `download`. The step of transcoding the media object when the object is downloaded could be completely transparent for the user of the object.

The actions associated to an operation can be defined statically or, alternatively, can be dependent on one or more variables. For example, a property of the media could be implemented as a variable. There are already solutions that address the particular case where the action to be taken depends on certain properties of the media (for example, format or size). Other important variables for defining the action are the context in which the object is being used and the media client itself. The ability to bind different actions to an operation, based on the particular format of the media or the state of the object in a workflow is a key aspect of a more dynamic SOA for media services. It allows you to define late binding between operations and actions taken on particular services. This approach works in synergy, together with the adoption of standardized ASDs and the adoption of MPEG-21 to represent digital media assets, to support the implementation of *abstract* workflows where services and operations are late-bound to the actual concrete services. The current implementation of Media Hub does not yet take advantage of DIP; however, these capabilities are being considered as requirements for future releases. Because the ASD ecosystem is based on the MPEG-21standard, it will be able to leverage these new features as soon as they become available.

[5] Digital Item Processing (DIP)

2

Media service ASD classes

We have identified the classes of media services that can be abstracted into a standardized interface definition. This following list of classes is not meant to be a comprehensive list but serves as a starter for identifying additional classes of media services:

- Transcoder: To perform conversion of one media format to another or to encode from analog source into digital media format
- Watermark: To embed a unique watermark in a digital media file
- Transport: To perform data movement of large media file from one server to another
- Repository (digital asset management): To add new asset, retrieve or manage existing digital asset of a catalog or repository.
- Media verification: To perform analysis or validate specific characteristics of the digital media file, normally as part of the quality assurance process
- Fingerprint: To perform forensic fingerprinting analysis of a digital media file
- Physical asset management (PAM): To add new asset or manage existing physical asset (for example, video or audio tapes) in the catalog or repository
- Resource scheduler: To schedule new or manage current available resources
- Publish: To perform packaging and publishing of the digital media to one or more distribution channels
- Edit: To help manage the Non-Linear Editing (NLE) process of a digital media
- Digital rights management (DRM): To manage media asset intellectual rights to help prevent illegal copying or conversation to other format

As mentioned in Chapter 1, “Abstract Service Definition overview” on page 1, common design patterns are applied across the different media service abstract definitions to help normalize and standardize the service interface. The design patterns are:

- Defined common terminology and naming convention used
- Use of MPEG-21 DID container to hold the media metadata in the request or response message structure
- Similar service, port types, operations, and message structure definition in the WSDL

- Same synchronous or asynchronous request and response interaction behavior with similar message XML structure used
- Use of same ACK response XML structure
- Defined common set of faults or errors used
- Defined common set of job or queue status used
- All ASDs have the status operations: manageJob and manageQueue
- All ASDs have two services defined: main service and client notification service
- All ASDs have the same number of port types defined: ReqRespWithNotification, Status, ReqResp, and Notification

In this chapter, the Transcoder abstract media service Web Services Description Language (WSDL) is used as an example to elaborate further about the previously discussed common design pattern or general features found across all the media service ASD classes. For further details about WSDL of the other media service ASD classes that have been defined thus far, see Appendix B, “Media Abstract Service Definition: WSDL” on page 47.

This chapter discusses the details of the Transcoder service ASD interface. As mentioned in Chapter 1, “Abstract Service Definition overview” on page 1, many aspects or general features of the media service interface specification are normalized or standardized among all the media service ASD classes.

Note: The Transcoder ASD specification discussed in subsequent sections is the latest draft, at this time of writing.

2.1 Transcoder service overview

A transcoder normally provides the capability to convert from one media format to another, to extract thumbnail images or closed captioned from a video, and to create a new video from one or more sources. The Transcoder ASD provides an abstraction layer to any third-party transcoding service as a new standard Web service interface for transcoding. This new transcoding Web service interface can be utilized in various business processes or be invoked by other service consumers who require a transcoding service.

The following high-level Transcoder Abstract Service functionalities can be exposed by the service provider:

- Transcode: To convert one media format to another
- Encode: To create a digital media file from an analog source (for example, tape or live data feed)
- Extract: To perform feature extraction such as thumbnail images or closed caption from a video.
- Conform: To embed graphics or to create a video from multiple sources.
- ManageJob: To perform Get Status, Cancel, Stop, and Change Priority of submitted jobs.
- ManageQueue: To get the current job list queued and status (length) of a job queue.

Ideally, the service provider has support for all or most of the these operations. However, as part of the ASD specification, a service provider does not have to implement all the operations. The service provider should implement, at the minimum, the Transcode and ManageJob operations to perform the main transcoding function and a way to query for status.

For an overview of the service operations, see the Transcoder ASD WSDL 2.2, “Transcoder ASD WSDL” on page 19.

2.2 Transcoder ASD WSDL

The Transcoder ASD Web service interface is described in a WSDL, as shown in Figure 2-1.

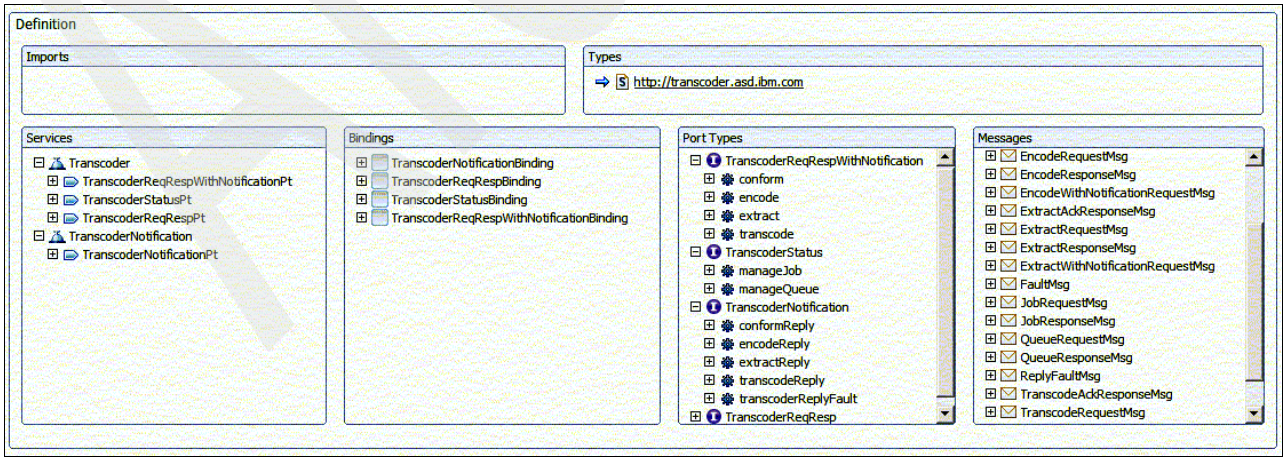

Figure 2-1 Transcoder Abstract Service WSDL

All of the media service ASD specification, including the Transcoder ASD WSDL, has two services defined:

- Transcoder service: Describes the interface for synchronous, asynchronous and managed status operations.
- TranscoderNotification service: Describes a client-implemented interface for asynchronous callbacks from the service provider to the invoking client to return the result of a specific operation.

The implementation of the Transcoder service operations as either synchronous or asynchronous depends on the integration pattern required for the solution. Usually, transcoding is performed as asynchronous because it is a long-running process that can take minutes to hours, depending on the media length and type of operation.

In general, the media abstract service WSDL has different port types defined, such as the following port types that are defined in the Transcoder service WSDL:

- TranscoderReqRespWithNotification: The service provider can implement the transcode service operations for this port as an asynchronous request with an immediate acknowledgement (ACK) response and then a client notification service callback when the job is completed.
- TranscoderStatus: The service provider can implement the job status and queue management operations.
- TranscoderNotification: The client can implement this port as its Web service notification listener, which receive client notification callbacks from the service provider where the service endpoint URI of this listener port is passed in the transcode request as the *replyTo* and *faultTo* element values for successful and failed jobs, respectively. See Figure 2-2 on page 21.
- TranscoderReqResp: The service provider can implement the transcode service operations using the request with a full response interaction pattern. This port should be implemented if the service provider wants to support synchronous or WS-Addressing asynchronous invocation style.

2.3 Transcode request structure

Figure 2-2 on page 21 shows the Transcoder ASD transcode with notification asynchronous operation request structure.

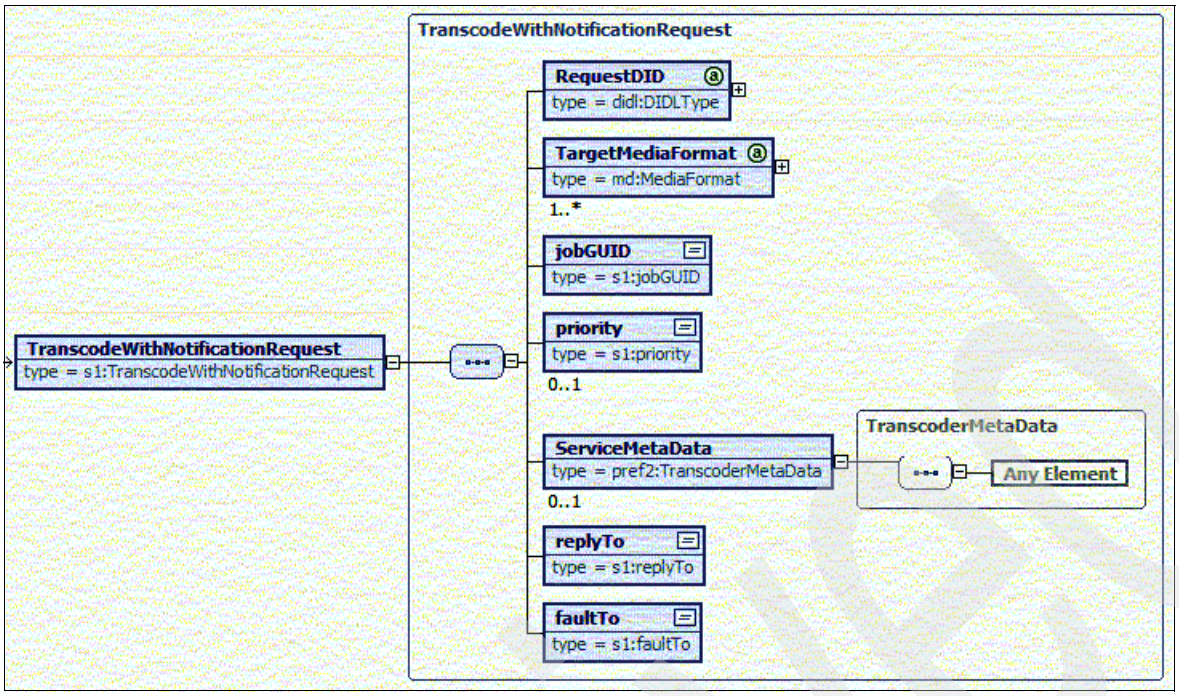

Figure 2-2 Transcoder ASD: TranscodeWithNotificationRequest structure

The transcode request structure consists of the following parameters:

RequestDID Indicates the MPEG-21 container that has the source media metadata and location. See 1.3.1, “General features of the ASD” on page 4 for further explanation.

TargetMediaFormat Specifies the output media format detail to transcode to. Figure 2-3 on page 22 shows the detail schema structure for TargetMediaFormat element.

jobGUID Indicates unique global job ID that client can pass in as reference or use as correlation. Normally, a vendor-specific job ID is created and used internally by the Transcoder.

priority Indicates priority setting that client can pass for a request. For example, setting can be low, normal, or high. Default is normal priority.

ServiceMetaData Describes the transcoder service provider specific metadata (for example, job profile name, output base filename, and more). This metadata is represented in a separate XML schema called TranscoderMetaData, which can be extended as appropriate to include the required additional metadata by the different service providers. (See 1.3.1, “General features of the ASD” on page 4 for further explanation.)

replyTo Indicates the client-provided service endpoint for the asynchronous callback from the service to the invoking client to return the result of an operation.

faultTo Indicates client-provided service endpoint for the asynchronous callback from the service to the invoking client to return fault or errors of an operation.

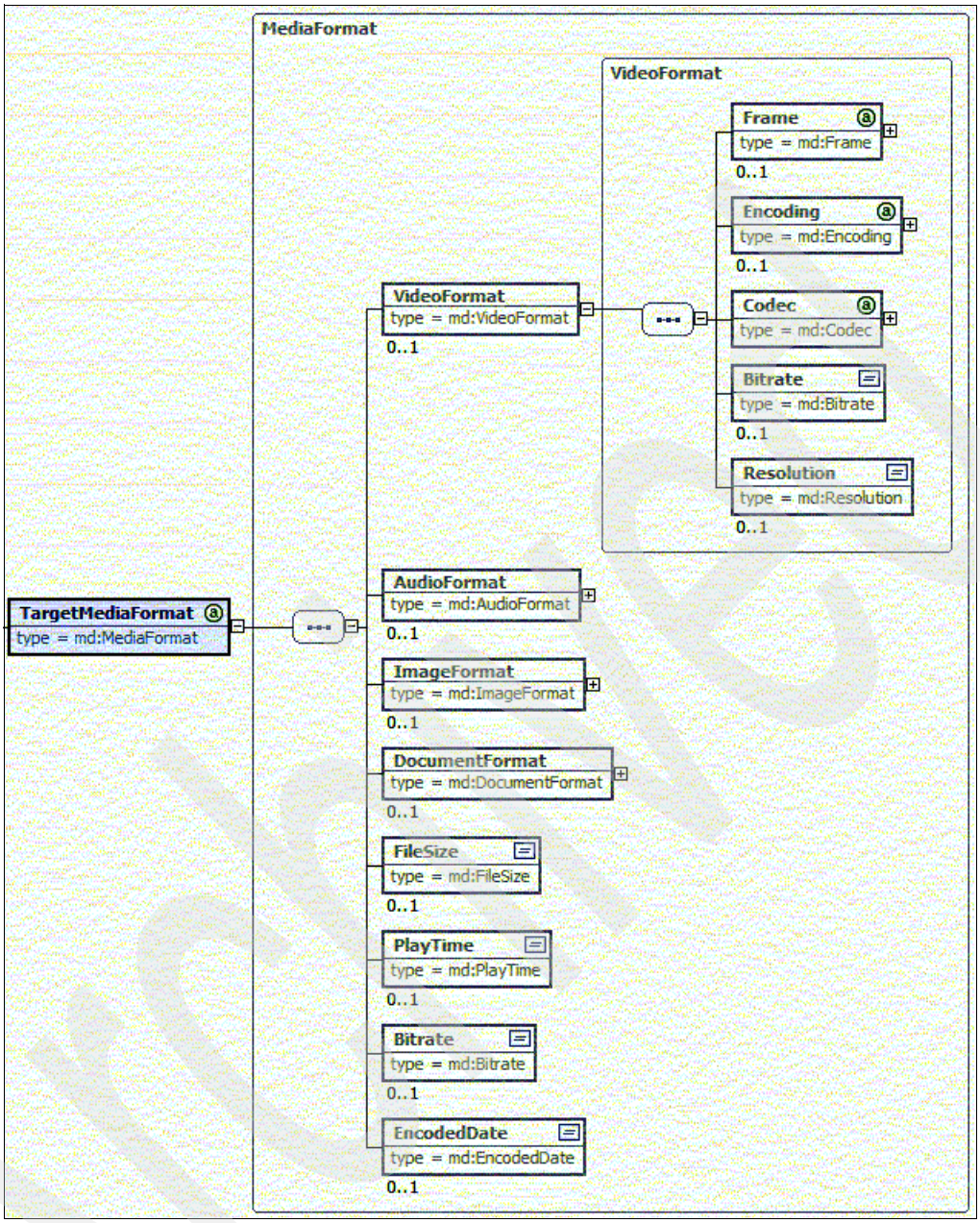

Figure 2-3 MediaFormat schema used for TargetMediaFormat element

2.4 Transcode ACK response structure

The Transcoder ASD supports an asynchronous transcode operations with notification. As previously discussed, this means that when the client calls the transcode service, the service return immediately with an acknowledgement (ACK) response, shown in Figure 2-4 on page 23. Then, at a later time, a callback from the service provider to the client notification service listener takes place to return the result of the operation.

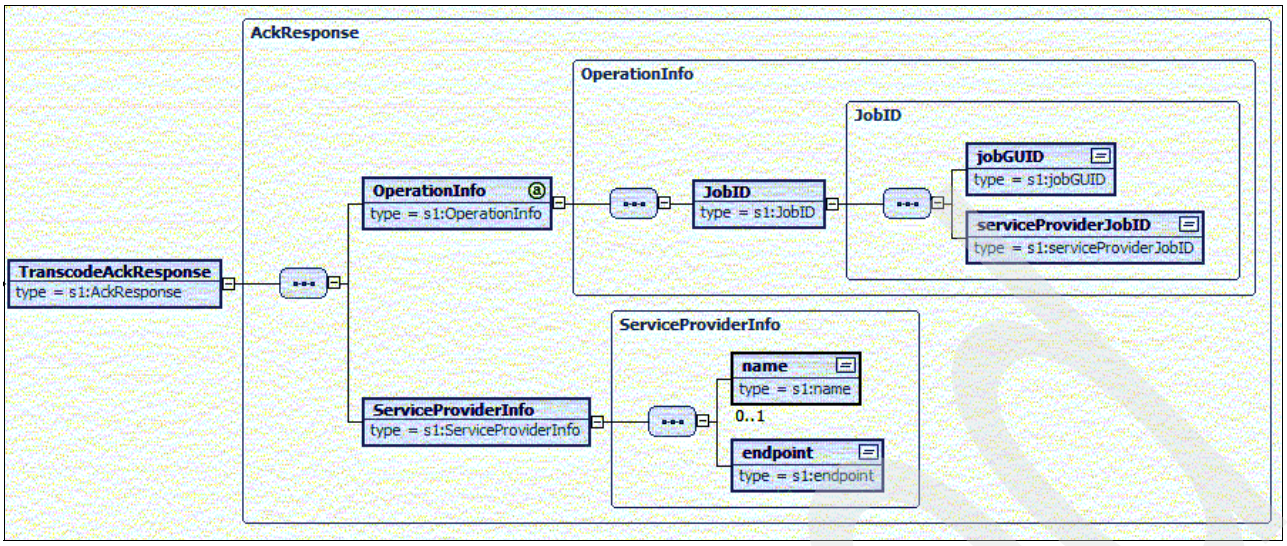

Figure 2-4 Transcoder ASD transcode ACK response structure

The *service ACK* response structure for asynchronous transcode with notification operations immediately returns the following information to the client requester:

OperationInfo Contains the service-provider-specific Job ID and the client's unique global job ID, if passed. The client may use these IDs for correlation to meet requirements.

ServiceProviderInfo Contains the service provider product name and a Web service endpoint information for the client to use in an IT dashboard or to directly invoke the service provider `manageJob` and `manageQueue` operations defined in the ASD specification to explicitly query for job status or manage the job queue.

2.5 Encode request structure

The Transcoder ASD provides an interface specification for those service providers that have the capability to perform encoding of an analog media source (such as video tape, audio tape or live data stream into digital media). Figure 2-5 on page 24 shows the Transcoder ASD encode with notification asynchronous operation request structure.

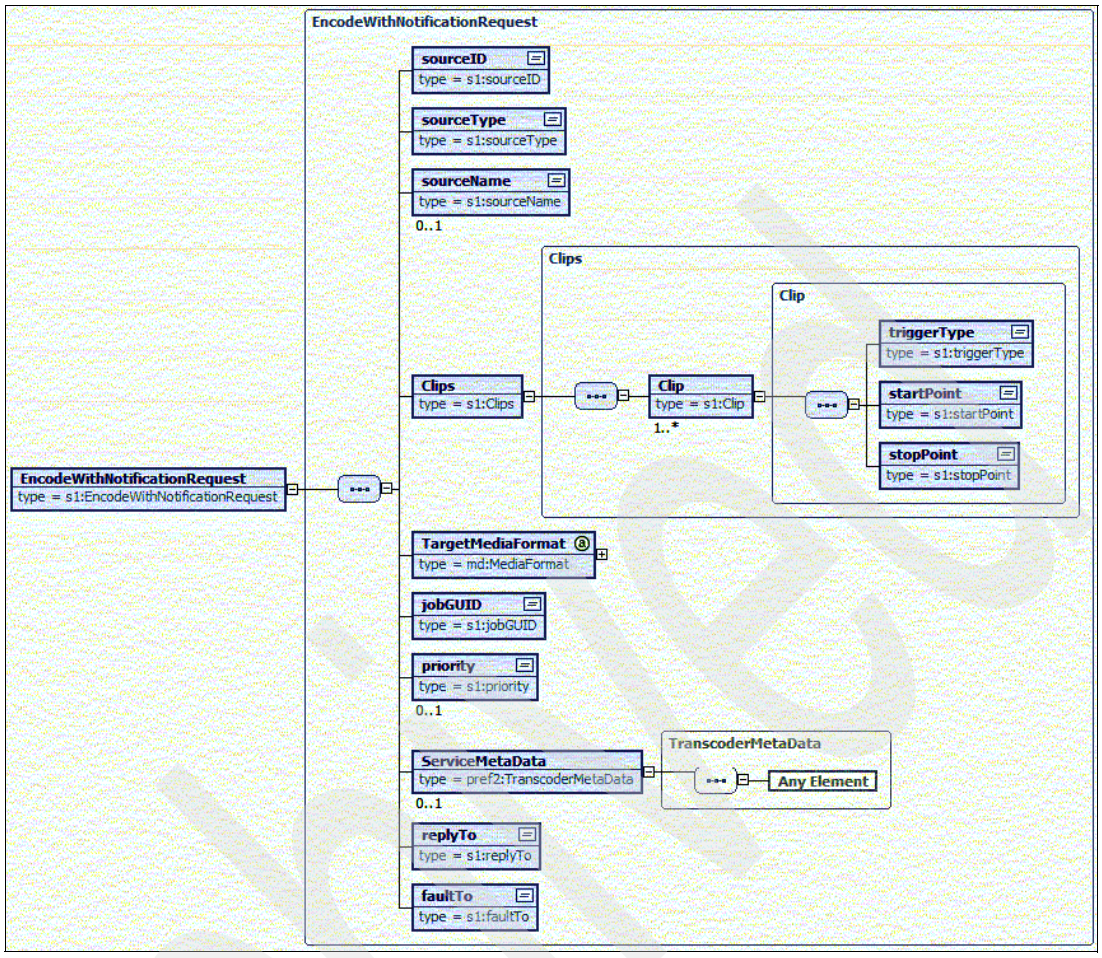

Figure 2-5 Transcoder ASD: EncodeWithNotificationRequest structure

The encode operation request structure is similar to the transcode request structure except for the following additional parameters:

sourceID Specifies a unique identifier for the analog input source device (such as a tape deck or live stream capture card) to be used for encoding

sourceType Specifies the type of analog source such as tape, DVD, live, and others

sourceName Specifies a descriptive name to be used when representing or displaying the analog input source device associated to the sourceID parameter

Clips Specifies one or more media clips to encode or digitize from the analog input source. A clip can be specified from beginning to the end or multiple parts of the analog source to be digitized.

The encode request ACK response is the same as the transcode operation.

2.6 Extract request structure

The Transcoder ASD provides an interface specification for those service providers that have the capability to perform data extraction from a media source, such as extracting thumbnail images, audio or closed captioned text from a video file. Figure 2-6 shows the Transcoder ASD extract asynchronous operation request structure.

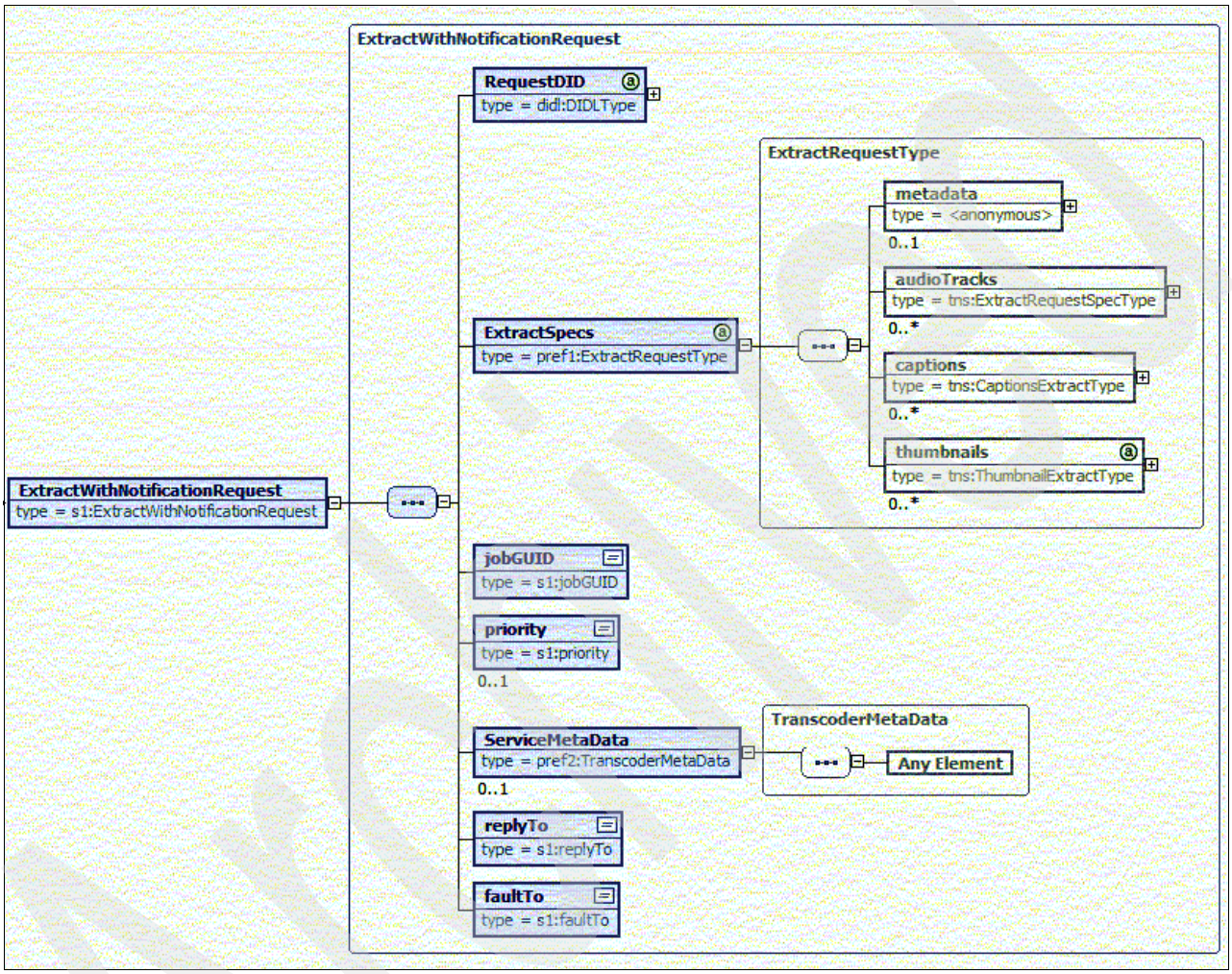

Figure 2-6 Transcoder ASD: ExtractWithNotificationRequest structure

The extract operation request structure is similar to the transcode request structure except for the additional parameter ExtractSpecs. The ExtractSpecs schema allows the client to specify what is to be extracted (for example, thumbnail image, audio, closed caption or metadata) and where in the source video time period to extract from.

The extract request ACK response is the same as the transcode operation.

2.7 Conform request structure

The Transcoder ASD provides an interface specification for those service providers that have the capability to embed text or image overlays and to conform multiple media source fragments or parts into a single complete media file, such as a video. Figure 2-7 shows the Transcoder ASD conform asynchronous with notification operation request structure.

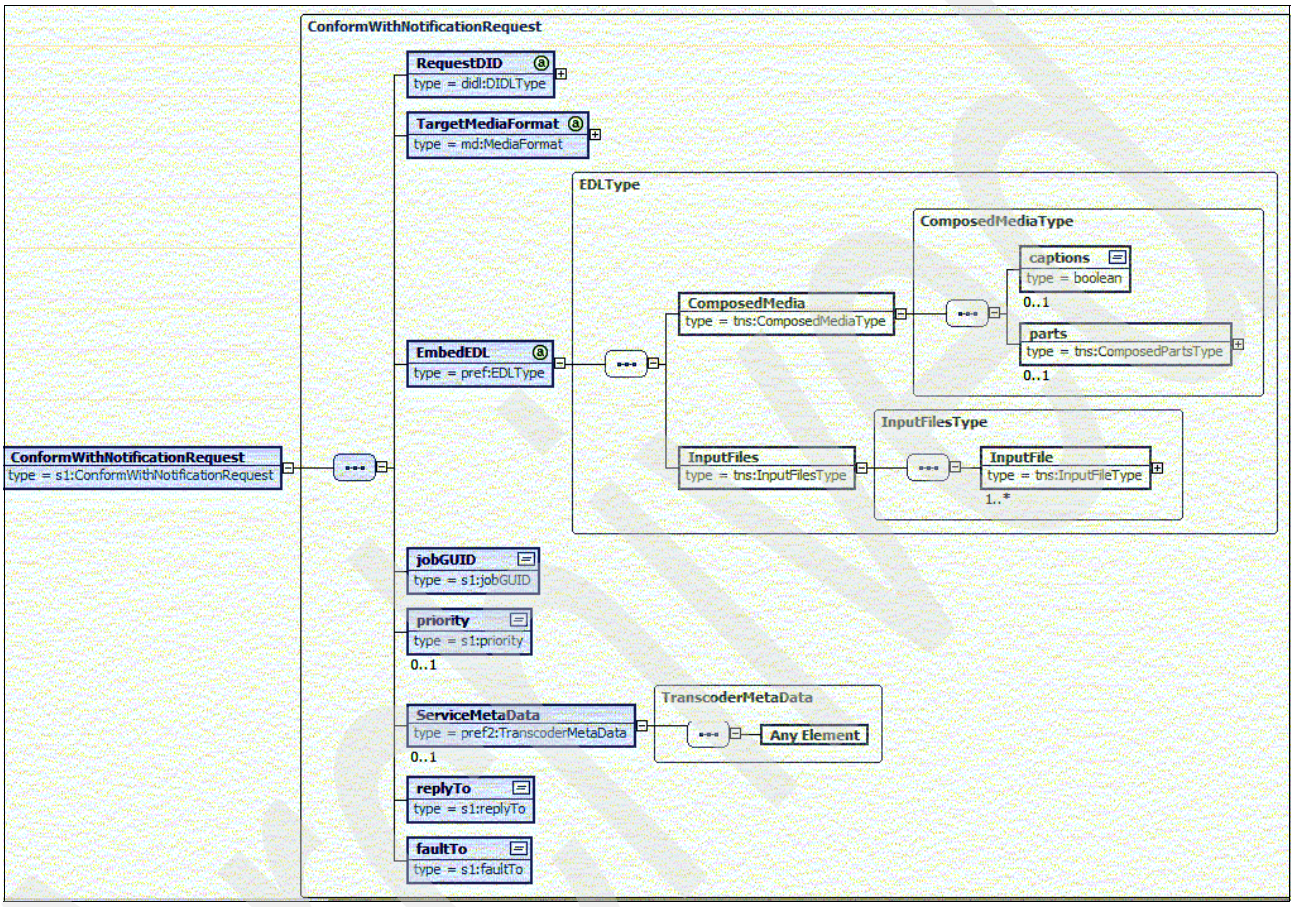

Figure 2-7 Transcoder ASD: ConformWithNotificationRequest structure

The conform operation request structure is similar to the transcode request structure except for the additional parameter EmbedEDL. The EmbedEDL schema allows the client to specify what is to be embedded or conformed (for example, thumbnail image, audio, closed caption or video fragment files) and where in the source media (for example, video time codes) to embed or overlay.

The ACK response to a conform operation request is the same as the transcode operation.

2.8 The manageJob request structure

The Transcoder ASD provides an interface specification for those service providers that allow the client to explicitly query and manage submitted transcode jobs. This is normally a synchronous operation. Figure 2-8 shows the Transcoder ASD manageJob operation request structure.

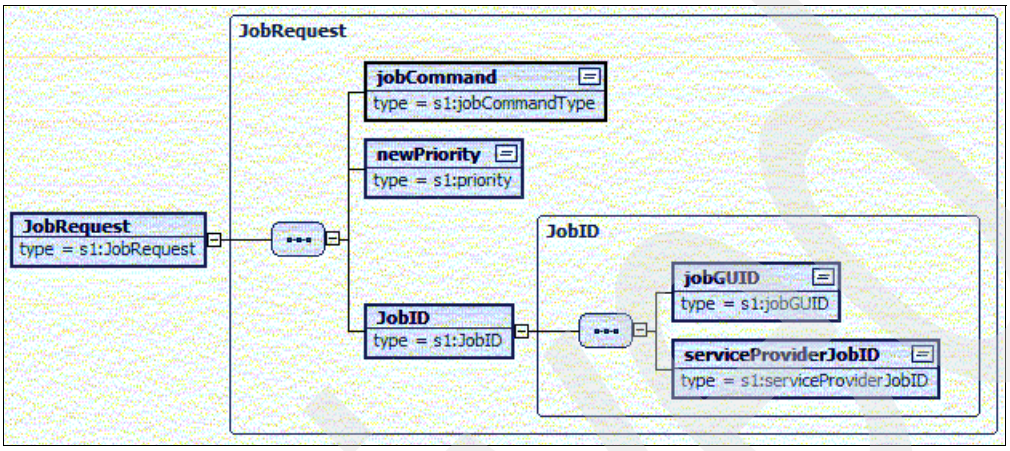

Figure 2-8 Transcode ASD manageJob request structure

The manageJob request structure has the following parameters:

jobCommand Specifies one of the following actions to take:

Status: Get job status detail
Cancel: Cancel or delete a job
Pause: Pause an active job
Stop: Stop an active job
Start: Restart a paused job
ChangePriority: Change priority of a job

newPriority Specifies the new priority setting for a job

jobID Specifies either the client's unique global job ID or a service provider's specific job ID for a job

2.9 The manageJob response structure

The response to a manageJob operation request is shown in Figure 2-9.

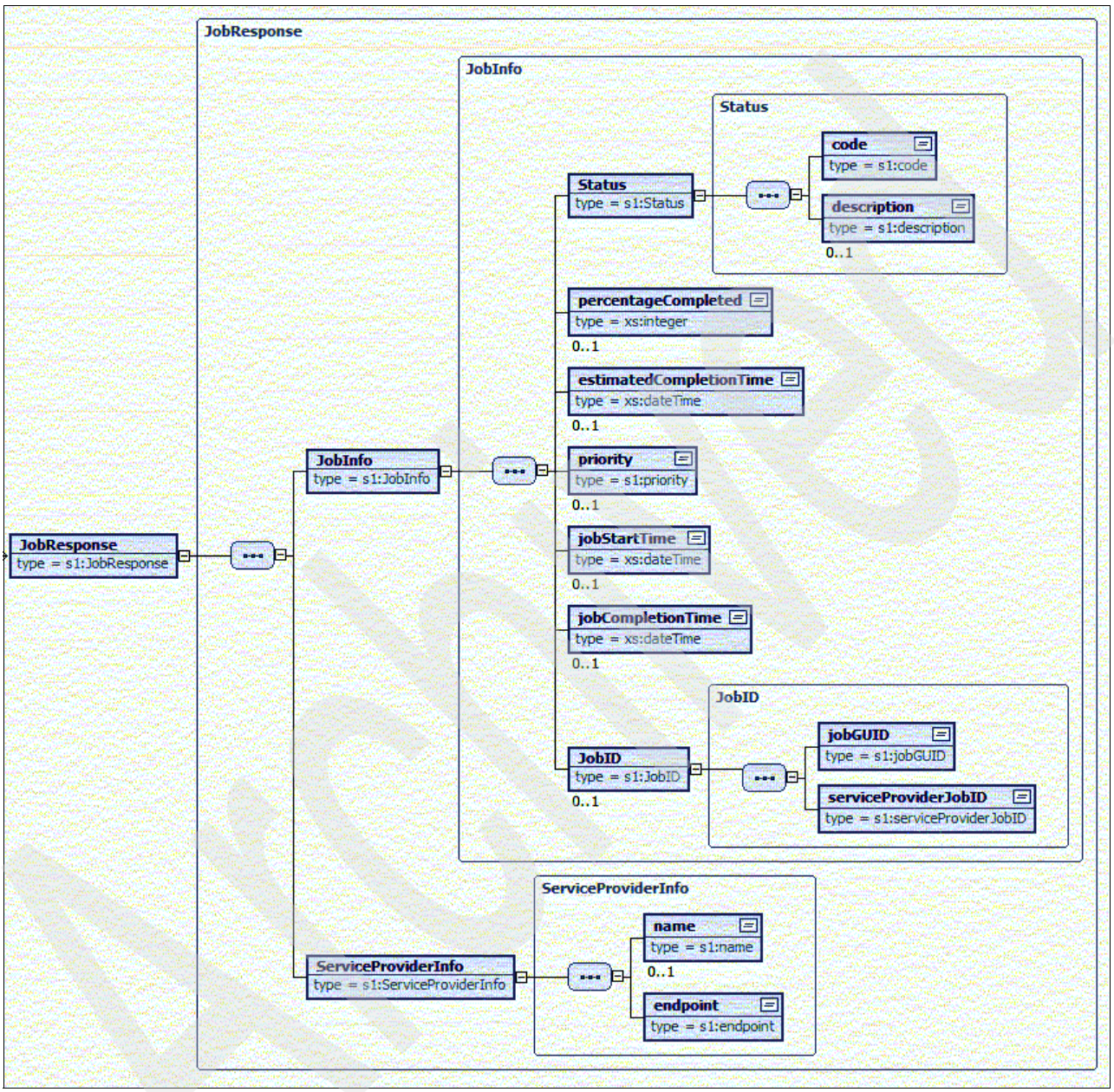

Figure 2-9 Transcoder ASD manageJob response structure

The manageJob operation response can include any of the following details about the requested job depending on the action requested:

status	Indicates current state of the job (see Table 2-1 on page 29)
percentageCompleted	Indicates percentage of completion of the running job
estimatedCompletionTime	Indicates estimated time of completion of the running job
priority	Indicates current job priority setting (low, normal or high)
jobStartTime	Indicates start time of the job

jobCompletedTime	Indicates completed time of job
jobGUID	Specifies client's unique global job ID
serviceProviderJobID	Specifies the job ID of service provider
ServiceProviderInfo	Contains the service provider product name and a Web service endpoint

Table 2-1 shows the job states defined and normalized across all the ASD media service classes where the job status code is returned.

Table 2-1 Job status definition

Job status	Description
READY	Job is currently in the queue and ready to start.
STARTED	Job has started.
PENDING	Job is currently in the process of applying the last job command such as canceling, stopping, pausing, and others.
COMPLETED	Job has ran to a completion successfully.
CANCELED	Job has been canceled successfully.
STOPPED	Job has been stopped successfully.
PAUSED	Job has been paused successfully.
FAILED	Job has ended with an error.
UNKNOWN	Job is in an unknown state.

2.10 The manageQueue request structure

The Transcoder ASD provides an interface specification for those service providers that support and allow the client to explicitly query and manage the service provider's job queue of submitted jobs. The assumption made for this operation is that it operates on a single logical queue facade interface, which can represent one or more physical queues managed by the service provider. The service provider is responsible to perform the mapping of a pending job to a physical queue and the aggregation view of a single logical queue. This operation is normally a synchronous operation. Figure 2-10 shows the Transcoder ASD manageQueue operation request structure.

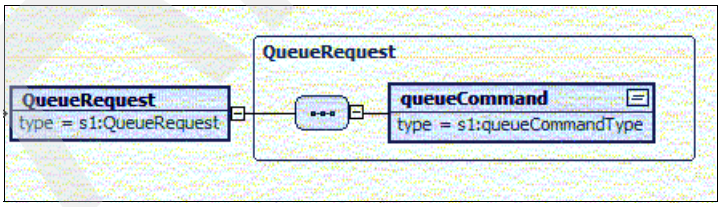

Figure 2-10 Transcoder ASD manageQueue request structure

The client request specifies one of the following actions as part of the queryCommand parameter:

- Status: Get status or current state of queue
- LockQueue: Lock a queue from receiving or accepting new job requests

- UnlockQueue: Unlock a locked queue
- ClearQueue: Remove or delete all remaining jobs in the queue
- List: List all jobs in the queue
- Stop: Stop or cancel all pending jobs in the queue from processing
- Start: Restart all stopped jobs in the queue

2.11 The manageQueue response structure

The service response to the client's manageQueue request is shown in Figure 2-11.

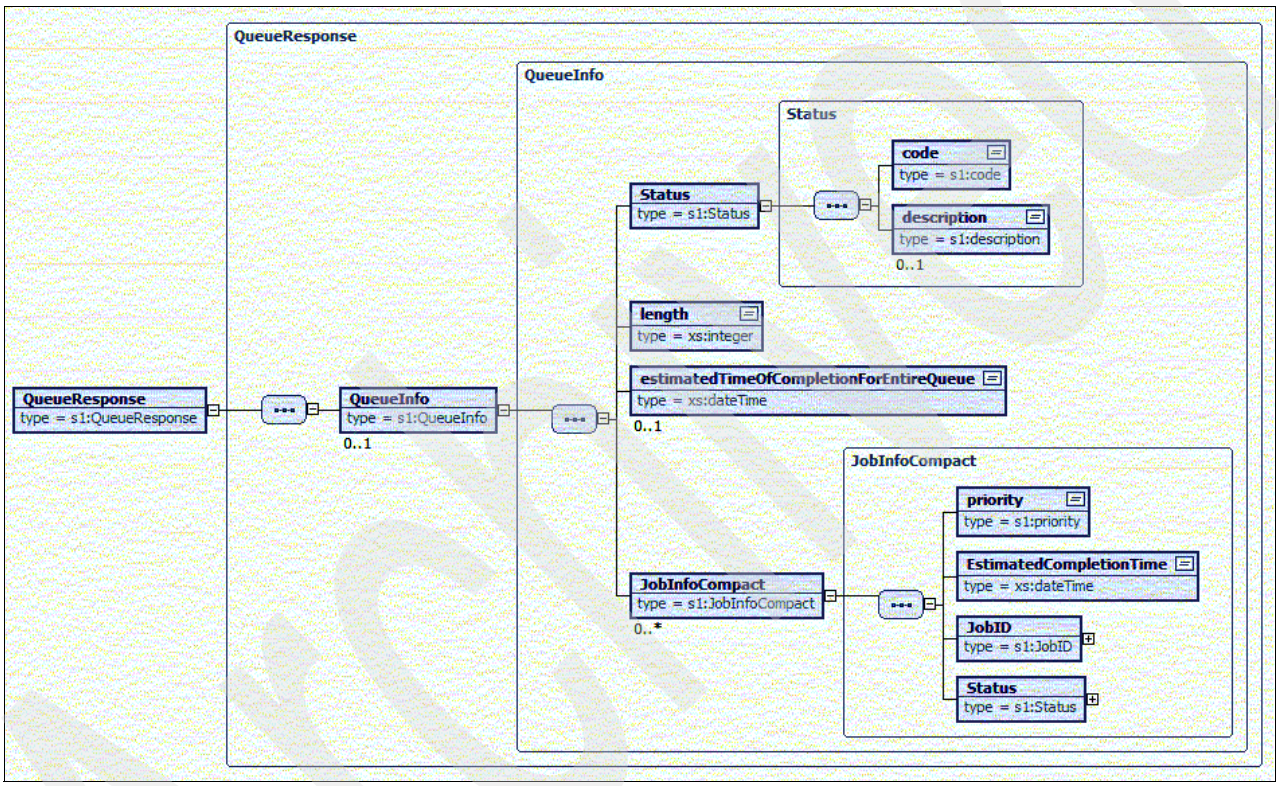

Figure 2-11 Transcoder ASD manageQueue response structure

The manageQueue response structure can include the following detail information depending on the queue command action requested:

- Current queue state or status (see Table 2-2 on page 31)
- Queue length or the total number of jobs queued
- Total estimated completion time for all the jobs in the queue
- List of jobs with associated job priority, job ID, status, and estimated completion time

Table 2-2 on page 31 shows the queue states defined and normalized across all the ASD media services classes when returning a queue status in the manageQueue response.

Table 2-2 Queue Status Definition

Queue status	Description
STARTED	The queue is running normally.
LOCKED	The queue has been locked successfully and is not accepting any new jobs submitted. The jobs already in the queue continue to start or run to completion.
STOPPED	The queue has been stopped successfully. The queue is not accepting new jobs submitted.
PENDING	The queue is in the process of applying the last queue command such as locking, clearing, unlocking, stopping, and others.
FAILED	The queue has failed with an error.
UNKNOWN	The queue is not running normally and is in an unknown state.

2.12 Full response structure

The Transcoder ASD full response structure shown in Figure 2-12 is used in the following use cases:

- Notification callback from the service provider to the client returning the result from the operation (such as transcode, extract, or conform)
- Synchronous request response
- WS-Addressing asynchronous request response

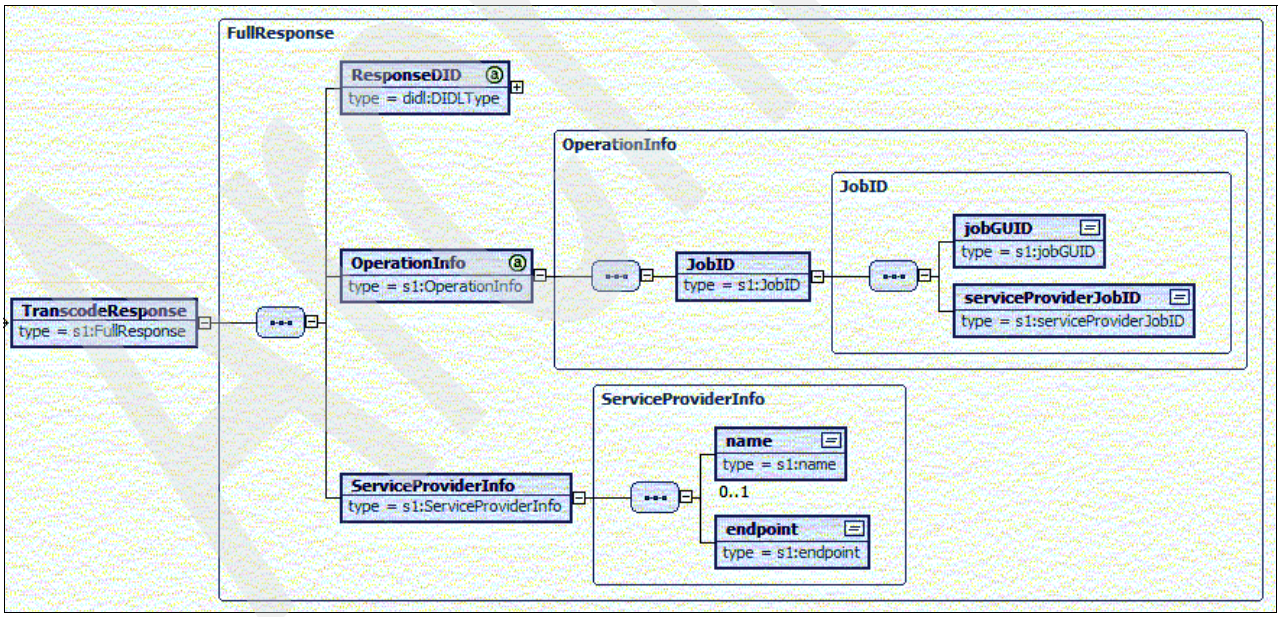

Figure 2-12 Transcoder ASD: FullResponse structure. for synchronous request and client notification callbacks

The full response structure returned to the client is similar to the transcode asynchronous request ACK response structure except for the addition of the responseDID MPEG-21 container, which has metadata that describes one or more returned output media files, depending on the requested operation.

2.13 Standard errors

One of the benefits in utilizing a standard media service abstract class is the inherit standardization of errors or in this case, SOAP faults. This benefit eliminates the necessity for the client requesters to implement different error handling logic for each individual service provider used. Now, the client only has to implement, once, a general compensation or error handling logic for all service providers that support a media service ASD class specification.

The standard errors are listed in Table 2-3. The table is not meant to be a comprehensive listing of all possible errors but is meant as an example to show only the different type of errors and their range of error codes, as follows:

- SYSxxxx: System level errors
- DATAxxxx: Data validation errors
- APPxxxx: Service level errors

Table 2-3 Standard errors

Error code	Error	Description
SYS0100	System Unavailable	Service is currently unavailable.
SYS0101	System Timeout	Adapter request has timed out before receiving a response.
...	...	(More system level errors)
DATA0100	Invalid Request XML Format	Request format was not valid (such as incorrect schema used or invalid XML).
DATA0101	Invalid Target Media Format	The target output media format specified is not valid.
DATA0102	Invalid Input Media Format	The input media format specified is not valid or not supported by this service.
...	...	(More request validation errors)
APP0100	Job Command is not currently supported	Job command is not currently supported by adapter or service provider.
APP0101	Queue Command is not currently supported	Queue command is not currently supported by adapter or service provider.
APP0102	Operation requested is not currently supported	Operation requested is not currently supported by adapter or service provider.
...	...	(More service level errors)

3

Guidelines for implementing an ASD-compliant adapter

This chapter provides general guidelines of how to implement an adapter that is compliant with an ASD interface.

We describe the steps necessary to create an adapter that implements an ASD interface.

3.1 Adapter objective

The objective of the adapter is to provide a functional interface at the appropriate abstraction level for the orchestration layer as discussed in Chapter 1, “Abstract Service Definition overview” on page 1. The adapter exposes the functionality provided by the vendor’s application to an ASD interface. An application can implement a much larger set of functionality than is defined at the ASD level. Only the functionality specified at the ASD level has to be managed by the adapter.

The adapter allows clients to call an operation that is defined in the ASD interface and receive a response as specified by the interface. The actual execution of the operation will mostly likely be implemented by a vendor’s application. In this scenario, a request from a client is received and interpreted by the adapter, manipulated in such a way that the vendor’s application can understand it, and then forwarded to the application. The application then executes the operation and sends a response to the adapter. The adapter interprets this response and generates a response to the client in accordance with the ASD interface. Figure 3-1 shows how the adapter abstracts the ISV application from the orchestration layer (client to the ASD adapter).

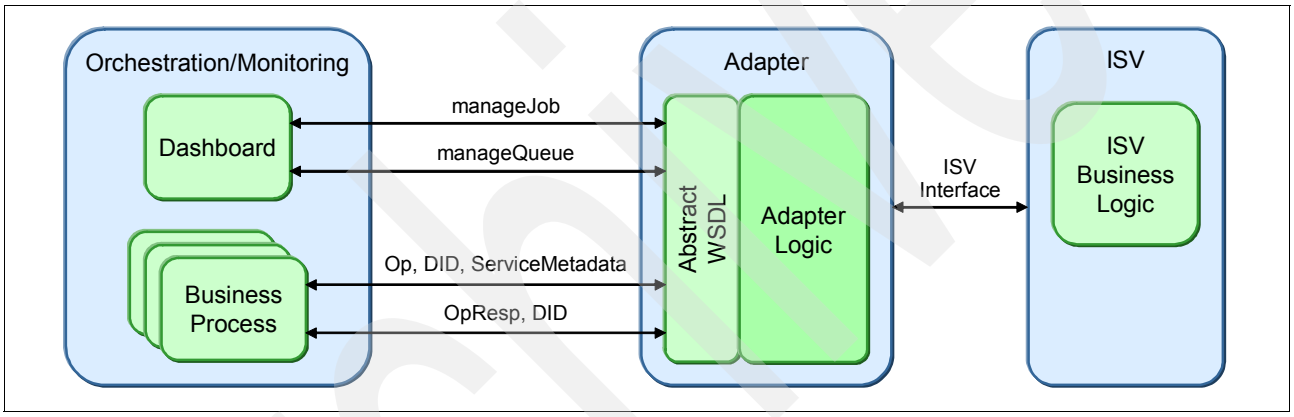

Figure 3-1 The Abstract adapter between the Orchestration/Monitoring and ISV systems

The adapter should provide the capability to receive asynchronous calls, allowing the client to make a request, and receive a callback when the operation completes at a much later time. It should keep track of all requests received and forwarded to the vendor’s application and be able to correlate the requests with the associated responses at both interfaces (ASD and vendor’s application).

One important aspect to consider is that each external application has its own way to specify properties that are defined in a more generic way at the ASD level. Although a property in an ASD might have the same semantic meaning of a property in a vendor application, it might use a completely different syntax to express it and might recognize only a set of values for the property. The ServiceMetadata element in the request message should carry these vendor-specific values when they are required to perform an operation. For example, the format of a media content is specified using an ID that is recognized in the ASD scope for a transcoding operation. However, at the application scope, a complete different way to represent the format might be employed. The adapter client should provide the format representation understood by the application as part of the ServiceMetadata, when invoking the service. Both format representations (ASD and vendor-specific) are included in the request message. The adapter response back to the client should always use the ASD representation.

3.2 Vendor's application

An ASD adapter most likely wraps an existing application to expose its functionality in compliance with an ASD interface. These applications can provide different ways to interact with it, such as Web services, Java, C#, Python, hot folder, and others.

The operations provided by the vendor might not map directly to the operations defined at an ASD interface. An operation at the ASD level might require multiple separate operations to be invoked on the vendor application. The adapter implementation is responsible to coordinate these calls and produce the result that is required by an ASD interface.

3.3 Adapter architecture

Figure 3-2 shows a generic architecture for an ASD adapter that covers the most common operations. This is by no means the only way to implement an ASD adapter and in many cases the implementation can be simpler than what is described here.

This architecture is based on reusable modules that facilitate the creation of other adapters. In the following sections, we describe these reusable modules.

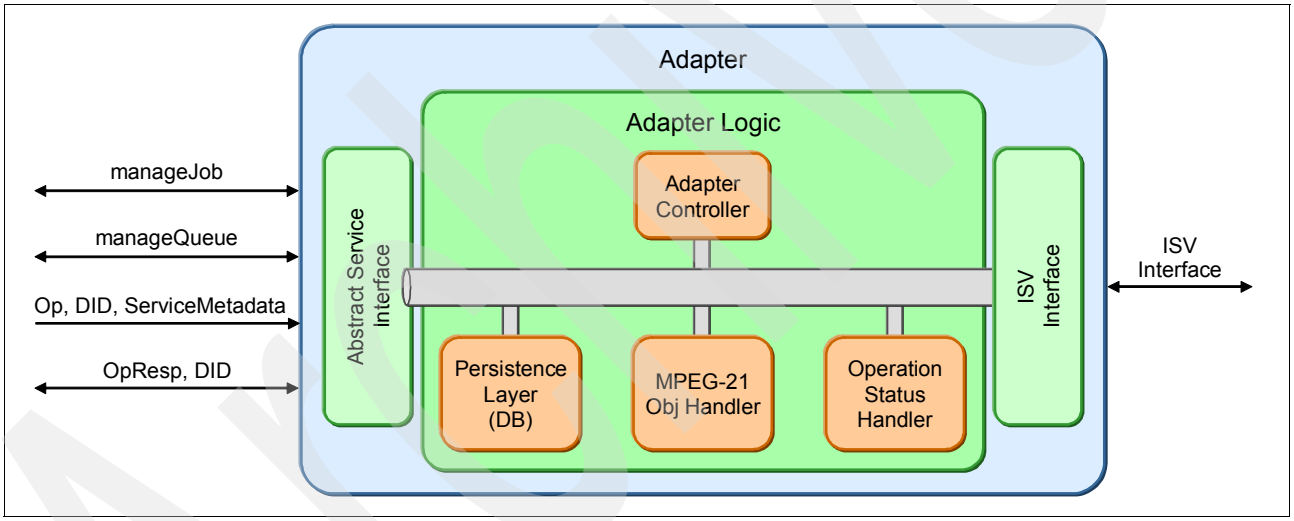

Figure 3-2 Adapter architecture

In Appendix A, "Sample code for Transcoder ASD adapter" on page 41, we describe an implementation of an adapter based on this architecture.

3.3.1 MPEG-21 module

This module is responsible to handle the MPEG-21 digital item that can be part of the request and response messages defined by an ASD. The MPEG-21 Digital Item Description Language (DIDL) schema defines the elements that can be present in a digital item. This schema is very flexible and allows for the many ways to represent a digital item. For the ASD interfaces, a specific representation, in terms of its internal elements, is required for an MPEG-21 digital item. This representation is described in "Putting it all together" on page 12.

The MPEG-21 module contains the functionality to extract the digital item from the request message (if present) and to include it in the response message (if required). The module

should provide a means to individually access the elements that are defined in the digital item. We suggest building an object representation of the digital item using a XML binding technology to interact with the digital item.

IDEs with Web services creation tools make the task easier. Such tools can take the WSDL and produce classes based on elements defined by its schemas. The elements of digital item are defined by the MPEG-21 Digital Item Declaration schema.

Two additional elements that are not part of the MPEG-21 schema are included in the digital item used by the ASD interfaces. They are the MediaDescriptor and the ResultMapList. The MPEG-21 schema defines an element `Statement` that can have as a child an element of `any type`. We use this place holder to add the two elements that are ASD specific into the digital item. These two elements are of *complex type* and have their own schemas to define their internal elements. Classes to manipulate the properties of the MediaDescriptor and ResultMapList can be produced from these schemas.

Various utility classes can be provided to facilitate the retrieval of the MediaDescriptor and the ResultMapList elements from a digital item as well as the creation of these elements in a digital item. The internal structure of the MediaDescriptor is presented in “Media Descriptor” on page 10, and the ResultMapList structure is described in “ResultMapList” on page 11.

Properties to be manipulated in the MediaDescriptor element include: file size, duration, creation date, bit rate, format ID, format name, format family, file extension, video codec attributes, and audio codec attributes. There is one MediaDescriptor per media contained in a digital item.

ResultMapList includes properties, such as source media URL and target media URL. ResultMapList describes the relationship between one or more source media with the derivatives (or target) media produced as the result of the operation. Only one ResultMapList exists in a digital item.

3.3.2 Persistence module

The persistence module provides the persistence layer for the adapter. This module is particularly useful for asynchronous operations because they usually include long-running processes, which can require that the operation status or context be persisted. It becomes important for error recovery capability on the adapter.

Basically, this module maintains the information of every operation request. This information is used for the processing of the requested operation as well as for tracking and monitoring of the operations performed by the adapter.

Information about a job request such as the request parameters, requester information, provider information, and job status are saved in a data store.

3.3.3 Operation status module

Certain categories of services (for example, transcoder) can support multiple jobs. Each job is associated to a client request to execute a specific operation (for example,. transcode). These jobs are usually en-queued and serviced in sequence. A priority mechanism can determine which one should be serviced next. This module provides the means to inquiry the application in regards to a particular job or queue, and take actions associated to these jobs and queues.

In cases where a vendor's application can only support one job at a time the adapter might be required to implement a queue structure internally to control the multiple requests it can receive. The operation status module will be then interfacing with this internal queue.

The ASD interfaces define a set of valid job and queue commands. They also define the valid status for jobs and queues. These standardized commands and status make it easier for the client to interact with the services because they enforce a common set of values.

3.3.4 Adapter controller

The adapter controller receives the request from an ASD client and coordinates its execution using the modules described previously.

Synchronous and asynchronous requests are handled differently by the controller.

Synchronous requests are processed in a single thread. The request is received by the controller and parsed to extract the parameters of the operation. A request to the vendor's application is then prepared, based on these parameters. Parameters that are specific to a particular vendor's application are passed through the ServiceMetadata element in the request message. The vendor's application is then invoked. The application executes the operation and returns the result as an object. The adapter then parses the resulting object and builds the response message to be returned to the client. Figure 3-3 contains pseudocode that shows the steps.

```
{ // Synchronous/Asynchronous (single thread) Operation Using Request/Response Port
  //  Parse message to retrieve request parameters
  parseRequest(message)
  //  Retrieve media information from digital item
  getMediaProperties(DigitalItem)
  //  Handle properties mapping  from ASD to ISV
  localMapping = mapInputParameters()
  //  invoke ISV operation and obtain result
  result = submitRequest()
   //  Handle properties mapping  from ISV to ASD
  abstractResult = mapResult(result)
   //  Build response message and return
  return (abstractResult)
}
```

Figure 3-3 Pseudocode for Synchronous/Asynchronous (single thread) implementation

A number of application servers support the WS-Addressing specification and enables transparent asynchronous interaction for Web services. This specification enables the execution of long-running operations without resulting in timeouts, because it decouples the Web services request/response session from the HTTP request/response session. The service request is associated to one HTTP session and the service response is carried by a separate HTTP session. The same code used to implement the synchronous interaction (as in Figure 3-3) can be used to implement an asynchronous interaction with WS-Addressing. This approach is possible because the WS-Addressing specification defines a number of message header elements that are processed by the application servers and gives the capability to handle the service request and response in separate HTTP sessions. Basically, the header carries information that tells the service where to send the response and also provides information to allow the client to correlate the request it issued with the response it received. All of this process is transparent to the code that implements the business logic of the service. So there is no difference, from the service provider perspective, if compared with the synchronous scenario, and everything continues to be handled by a single thread. At the

end of the operation execution, the application server opens a connection to the endpoint specified in the message header and sends the response message automatically. The difference is really in the way the client issues the request and receives the response. In the synchronous case, the client issues the request and expects the response of the operation in the same HTTP connection. In the asynchronous scenario, the client issues the request with the proper WS-Addressing fields in the message header and expects the response as a callback from the service provider to the client. The client should implement a service to receive the callback from the service provider.

Another approach to support asynchronous interactions is to separate the request and response processing threads. The first thread receives the request from a client, initiates the job processing, and returns a message to the client to acknowledge that the request has been received. The second thread sends the result of the processing to the client after the job execution is completed. This process requires two distinct port types, one to receive the request and another to send the response (callback). This is the best approach to handle very long-running processes (that can take up to hours, days, or even longer to run).

All ASD interfaces provide port types to implement the two mechanisms for asynchronous services.

In the latter asynchronous scenario, the request is received by the controller, processed by the adapter, and the vendor application is invoked to perform the requested operation. The application does not return with the result immediately. Instead, the application can return local information that identifies the particular request. The information about the original request, including the client callback endpoint along with the local information from the application, is persisted in a data store. The final response message is not to be produced by this thread. It will be generated by a separate thread when the request processing is finalized. This part is the first half of this asynchronous scenario processing and can be implemented in a single thread. A pseudocode of these steps is shown in Figure 3-4.

```
{ // Asynchronous (independent threads) Operation Using Request and Callback Ports
  //  Parse message to retrieve request parameters
  parseRequest(message)
  //  Retrieve media information from digital item
  extract(DigitalItem)
  //  Handle properties mapping  from ASD to ISV
  localMapping = mapInputParameters()
  //  invoke ISV operation and obtain handle
  localJobInfo = submitRequest()
  //   Persist this request with relevant information to datastore
  persistOperation(requestMessage, localJobID, localMapping, status)
  //   Build ack message with localJobInfo
  return (ack)
  }
```

Figure 3-4 Pseudocode for Asynchronous (independent threads) implementation

The second part of the asynchronous process is handled by a polling thread. A pseudocode that implements this thread is shown in Figure 3-5. The thread checks and retrieves a list of active requests by inquiring the data store periodically. It then invokes the vendor's application to find the current status of each one of these requests. If the status indicates that the operation has completed successfully, the result of the operation is retrieved. This resulting data is then parsed and, together with job data retrieved from the database, is used to build the response message to be sent to the client. This message contains parameters that allow the client to correlate this response with a particular request that it issued previously. The response message is sent to the reply-endpoint specified in the original request message. The data store is then updated to indicate that the operation has completed successfully.

```
{ // Polling Thread Module
  //  Obtain list of all active requests from datastore
  listActiveRequest = PersistOperation(listActiveRequest)
  //  For each active request
  iterate on listActiveRequest with I
    { // Request job status from the ISV
      status = getStatus(listActiveRequest[i])
      //  Check status
      switch (status)
        //  Job completed successfully
        case sucess:
        { // Retrieve operation result from ISV
          result = getJobResult(listActiveRequest[i])
          //  Handle properties mapping  from ISV to ASD
          abstractMapping = mapResult(result)
          //  Build response message and send to client
          sendResponseMessage(abstractMapping)
          //  Update datastore status for this request
          persistOperation(listActiveRequest[i], sucess)
        }
        //  Job still in process
        case active:
        {   //  Check for operation timeout
            if (handleTimeOut(listActiveRequest[i])
            {  // Operation timed out
             //  Prepare fault message and send to client
             sendFaultMessage(abstractMapping)
             //  Update datastore status for this request
             persistOperation(listActiveRequest[i], timeOut)
             }
             // Else, do nothing
        }
        //  Job failed
        case fail:
        { //  Get fault details from ISV
          fault = getFault(listActiveRequest[i])
          //  Handle properties mapping  from ISV to ASD
          abstractMapping = mapResult(fault)
          //  Prepare fault message and send to client
          sendFaultMessage(abstractMapping)
          //  Update datastore status for this request
          persistOperation(listActiveRequest[i], fault)
        }
    }
  }
```

Figure 3-5 Pseudocode for Polling Thread

If the inquiry for the status of an active request indicates that an error occurred during its execution, a request to retrieve the error details is issued to the application. The error-details

information is then processed to produce an error code that is defined by the ASD interface. The ASD interface defines a set of standardized error codes. Then a fault message carrying the fault details is sent back to the endpoint defined in the original request message. The data store is updated to indicate the request did not complete successfully.

If the application indicates that the request is still being processed, the elapsed time (since the request was issued to the application) is checked to verify whether an allowed period of time for the execution has expired. If not, no further processing is required. If a timeout is observed, a fault message is sent to the endpoint indicated in the request message, and the data store is updated with error status. The particular request then must be cancelled on the application side (to be added in the pseudocode).

Several vendor applications offer the possibility to notify the requester (in this case the adapter controller) when the job execution completes. In this particular scenario, querying the vendor's application to check for the status of a job is unnecessary. The adapter controller would have to provide a listener to receive the notifications. This notification would most likely carry the result of the operation also, so explicitly requesting it from the application is unnecessary.

A

Sample code for Transcoder ASD adapter

This appendix describes the implementation of a sample adapter that complies with the ASD interface for transcoders.

This sample code is available as a downloadable .zip file. See Appendix D, “Additional material” on page 57 for details.

Description

This sample implementation wraps a sample transcoder application and expose it through the Transcoder ASD interface, for usage in orchestrated business processes. The sample transcoder application in this example is accessed through a Java interface. The adapter uses this interface to communicate with the application.

The adapter implements all the components defined by the architecture shown in Figure 3-2 on page 35. See Chapter 3, “Guidelines for implementing an ASD-compliant adapter” on page 33 to understand all the components involved and how they interact with each other to provide the required functionality.

An Apache Derby database was chosen to provide the persistence layer.

A mapping service provides the format mapping functionality between the ASD interface and the sample transcoder application.

This sample code implements only the transcode operation as defined in the Transcoder ASD WSDL. By following the sample code, one can very easily implement the business logic for the other operations. Both the synchronous and the asynchronous (with WS-Addressing and notification callback) interactions are supported. The asynchronous scenario with notification includes a database and allows for the request and response messages to be handled by separate port types.

Prerequisites

The sample code was developed with the following software:

- Eclipse IDE for Java EE Developers 3.4.1 (Ganymede), which is available at:

  http://www.eclipse.org/downloads
- Apache Tomcat 6.0.18, which can be downloaded from:

  http://tomcat.apache.org/download-60.cgi
- A set of Eclipse plug-ins for Derby database:
  - derby_core_plugin_10.4.2.zip
  - derby_ui_plugin_1.1.2.zip

  They can be obtained from:

  http://db.apache.org/derby/derby_downloads.html
- Axis2 1.4.1, which is located at:

  http://ws.apache.org/axis2/download/1_4_1/download.cgi
- JRE™ 6.0, which is located at:

  http://java.sun.com/javase/downloads/index.jsp

Install Eclipse with Tomcat as a server. Install the Derby plug-ins. The article, located at the following Web address, shows how to configure Eclipse to work with Derby:

http://www.eclipse.org/articles/article.php?file=Article-EclipseDbWebapps/index.html

Included projects

The sample code in the distribution includes the following projects, which are described in more detail in subsequent sections:

- TranscoderAdapterAxis2: A Web project with the implementation of the service. This Web service is implemented using Axis2. This project implements the module described in 3.3.4, “Adapter controller” on page 37. It also implements the job management functionality as defined in 3.3.3, “Operation status module” on page 36.
- SampleTranscoder: A Java project with a simple example of an application that provides transcoding functionality. This application provides a Java API to initiate a transcoding operation, check for its completion, and retrieve the operation result. This implements the application defined in 3.2, “Vendor’s application” on page 35.
- ASDUtilsAxis2: A Java project with a helper class to facilitate the manipulation of elements that are carried in the MPEG-21 digital item. It is not part of its schema, namely the MediaDescriptor and the ResultMapList. It maps to the module defined in 3.3.1, “MPEG-21 module” on page 35.
- data: A Java project that holds the Derby database for this sample code. The Derby plug-in uses this project as the repository for the database in the Eclipse workspace, which provides the functionality of the module specified in 3.3.2, “Persistence module” on page 36.

The TranscoderAdapterAxis2 project

The Web service was created with the Eclipse Web services plug-in using the Transcoder ASD WSDL as the starting point (top-down approach). For this sample code purpose only, a subset of the defined operations have been implemented:

- The `transcode` operation is defined in the TranscoderReqResp and TranscoderReqRespWithNotification port types.
- The `transcodeReply` and `transcoderReplyFault` operations are defined in TranscoderNotification port type.
- The `manageJob` operation is defined in the TranscoderStatus port type.

TranscoderReqResp, TranscoderReqRespWithNotification, and TranscoderStatus port types are implemented as Web service provider. And the TranscoderStatus port type is implemented as Web service client. Because the Eclipse Axis2 plug-in tool can handle only one port type for code generation, the original WSDL was subdivided to produce one WSDL per port type. The WSDLs that were used to create a service are in the following location:

`/WebContent/wsdl/`

This sample Web service is implemented by using Apache Axis2 as the runtime. By using the TranscoderReqResp port type, both interactions (synchronous and asynchronous) can be supported as mentioned in Chapter 3, “Guidelines for implementing an ASD-compliant adapter” on page 33. The type of interaction depends on how the client invokes the service. If WS-Addressing headers are present in the request message, the adapter will handle the request asynchronously, otherwise the request will be processed synchronously.

The adapter also supports the processing of the request asynchronously in separate threads for request and response. In this case, the TranscoderReqRespWithNotification port type is used for the request, and the TranscoderNotification port type (with the `transcodeReply` operation) is used for the response. The `transcoderReplyFault` operation of the TranscoderNotification port type is used if an error is detected in the latter part of the

processing of the operation. If an error is detected upon the reception of the request, a fault message is issued through the TranscoderReqRespWithNotification port type.

The `manageJob` operation of the TranscoderStatus port type allows for the adapter to take actions on a job submitted previously. Because of the limited functionality of the sample transcoder application, only the actions to request a status and to cancel a job are supported.

Using the Eclipse Web services plug-in to create the service from the WSDL, a number of classes distributed in a set of packages are produced. Only the skeleton classes (TranscoderReqRespSkeleton, TranscoderReqRespWithNotificationSkeleton, and TranscoderStatusSkeleton in this sample) has to be edited to include the business logic associated to the supported operations.

TranscoderReqRespSkeleton should contain the logic to handle the synchronous and single thread asynchronous interaction.

TranscoderReqRespWithNotificationSkeleton implements the two-thread (request/callback) asynchronous interaction. The information about the request, along with parameters that identify the initiated job at the application side and how to notify the requester when the job is finalized at a later time, is saved in the Derby database by this skeleton class.

A continuously running thread (implemented by the DBJobManager servlet, which is initialized at the server startup), is used to poll the database periodically and search for requests that are still active (meaning the transcoder application was still processing it the last time it was checked). For every one of these requests, the transcoder application is invoked to check for the request's current status. If the request is completed, the result of the job is retrieved and the client is called back with the result of the operation. If the transcoder application indicates an error occurred during the processing of the request or it timed out (the process did not complete in a specified time), the client is notified with the error details. If the request is still running, nothing is done. In each case, the database is updated accordingly.

TranscoderStatusSkeleton implements the logic to interface with the sample transcoder application to manage job requests previously issued through the two skeleton classes mentioned previously.

A properties file (`config.properties`) under `/com/ibm/asd/config/` contains the configuration properties used by the adapter.

The SampleTranscoder project

This Java project simulates a transcoder application. It does not provide any real transcoding capability. The objective is to emulate a transcoder application that provides a Java interface to demonstrate how to implement an adapter. This interface provides methods to request the transcoding of a media, to check the status of a request and retrieve its result.

The interface is defined by `SampleTranscoder.java` and the application main class is `SampleTranscoderImpl.java`. For the sake of simplicity, this class can handle only a single request at a time. A new request will always override the current one. A more realistic implementation would support multiple simultaneous requests.

In this sample application, a request (or job) is considered completed after a number of inquiries are made about the status of the job. This number is configurable and defined in a properties file (`config.properties`) under `/sample/transcoder/config/`.

After a request is completed, a Result object with the information about the transcoded media can be retrieved by using the `getResult()` operation.

The ASDUtilsAxis2 project

All the artifacts necessary to handle the MPEG-21 digital item, as an object with getter and setters for all its sub-components, are generated by the Eclipse plug-in when creating the Web service from the WSDL. However, the two elements that are carried in the digital item and that are not part of MPEG-21 schema are the MediaDescriptor and the ResultMapList. These elements are inserted in fields that are defined as `any type` in the MPEG-21 schema, and are represented by XML documents.

The generated code handles these elements as `org.apache.axis2.om.OMElement` and does not give direct access to their internal structures.

The ASDUtilsAxis2 Java project provides a utility class (`Binding.java`) to allow the transformation of these elements between their `OMElement` and Java representations. This way makes the handling of these elements and their internal structures much easier, and assumes you are using Axis2 as the Web services runtime for the adapter project.

The data project

In this sample implementation, we use an Eclipse plug-in to deal with the Derby database. This approach helps to more easily create and maintain a database in a development environment. Using this plug-in implies that a Java project will maintain the Derby databases to be used in a Eclipse workspace.

This project functions as the repository for the database used by the sample adapter. The database name used by the sample adapter is `ASDAdapterDB`.

B

Media Abstract Service Definition: WSDL

This appendix shows a high-level overview of the WSDL for Transcoder, Watermark, Transport, Repository and Media Verification abstract media services. These WSDL and associated XSDs file can be downloaded. See Appendix D, “Additional material” on page 57.

Note: All specifications discussed here are the latest drafts at this time of writing.

Specifications discussed include:

- Transcoder Abstract Service WSDL
- Watermark Abstract Service WSDL
- Transport Abstract Service WSDL
- Repository Abstract Service WSDL
- Media Verification Abstract Service WSDL

Transcoder Abstract Service WSDL

A transcoder normally provides the capability to convert from one media format to another, to extract thumbnail images or closed captioned from a video, and to create a new video from one or more sources. The Transcoder ASD provides an abstraction layer to any third-party transcoding service as a new standard Web service interface for transcoding. This new transcoding Web service interface can be utilize in various business processes or be invoked by other service consumers requiring a transcoding service.

The following high-level Transcoder Abstract Service operations can be exposed by the transcoder service provider:

- Transcode: To convert from one media format to another
- Encode: To create a digital media file from an analog source (for example, tape or live data feed)
- Extract: To do feature extraction, such as thumbnail images or closed caption from a video
- Conform: To embed a graphic or to create a video from multiple sources
- Manage Job: To get status, cancel, stop and change priority of submitted jobs
- Manage Queue: To get current job list queued, length and status of a job queue

These operations are shown in the WSDL in Figure B-1.

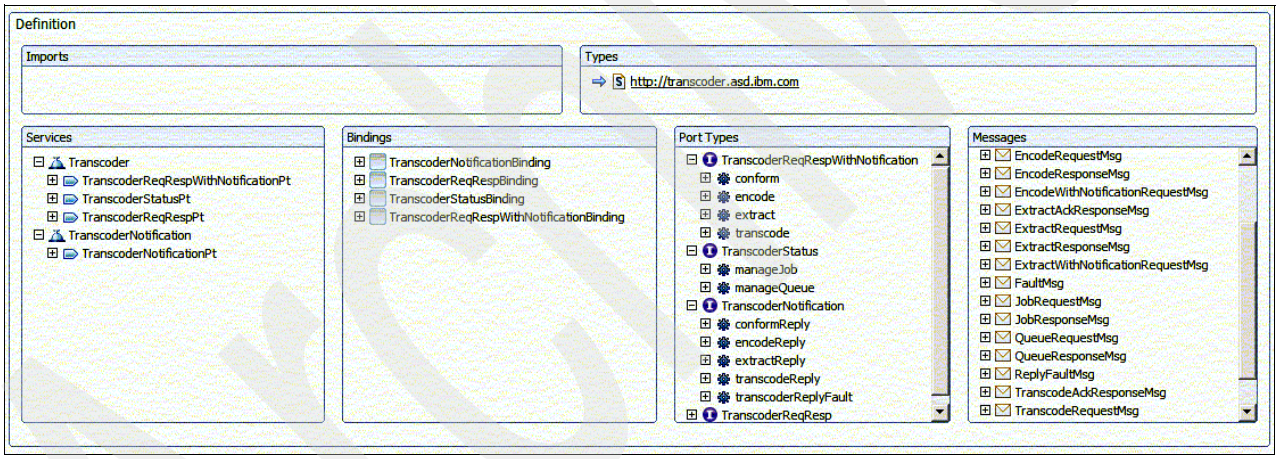

Figure B-1 Transcoder Abstract Service WSDL

See 2.1, “Transcoder service overview” on page 19 for details about the Transcoder ASD.

Watermark Abstract Service WSDL

A watermark service provides the capability to embed a unique watermark identifier as either a graphical image or text into a media file (for example, a photo, a graphical image, a video, and others). The Watermark ASD provides an abstraction layer to any third-party watermarking service as a new standard Web service interface for watermarking.

The following high-level Watermark Abstract Service operations can be exposed by the watermark service provider:

- Embed Watermark: To create a new watermark for a media file
- Get Embed Info: To retrieve the metadata associated with the watermark in the media file
- Retrieve Watermark ID: To extract and retrieve the embedded watermark in the media file
- Manage Job: To get status, cancel, stop and change priority of submitted jobs
- Manage Queue: To get current job list queued and status (length) of a job queue

These operations are shown in the WSDL in Figure B-2.

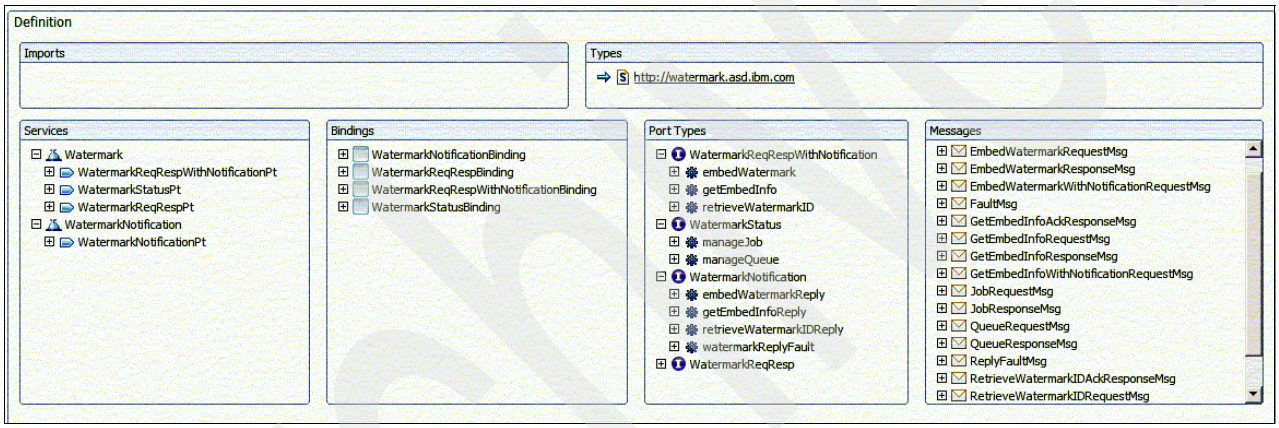

Figure B-2 Watermark Abstract Service WSDL

Transport Abstract Service WSDL

A transport or data mover service provides the capability to copy and physically transfer large media files from one server to another through proprietary methods. The Transport ASD provides an abstraction layer to any third-party transport or data mover service as a new standard Web service interface for digital file transfer.

The following high-level Transport Abstract Service operations can be exposed by the transport or data mover service provider:

- Transfer Assets: To copy and transfer media files from one server to another
- Manage Job: To get status, cancel, stop and change priority of submitted jobs
- Manage Queue: To get current job list queued and status (length) of a job queue

These operations are shown in the WSDL in Figure B-3.

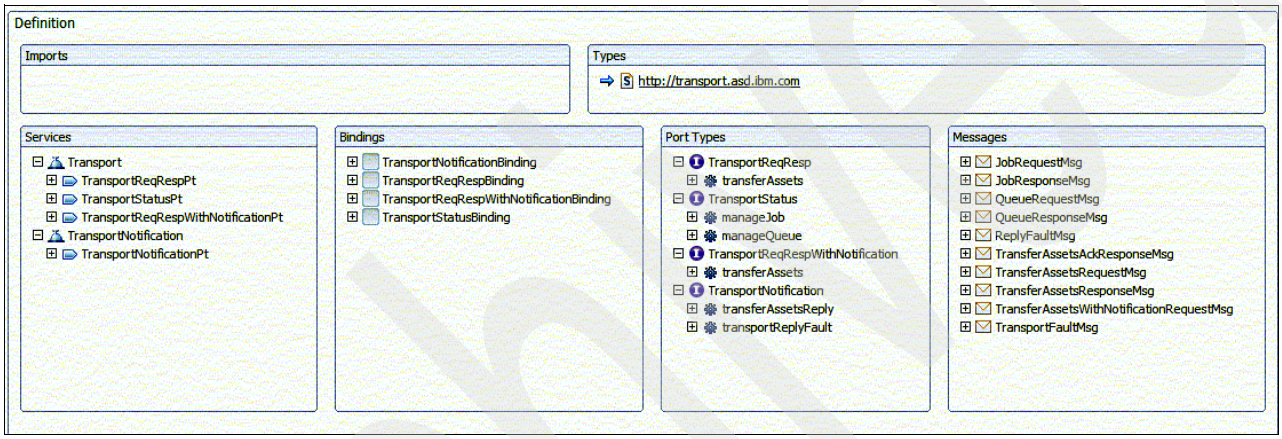

Figure B-3 Transport Abstract Service WSDL

Repository Abstract Service WSDL

A repository or digital asset management (DAM) service provides the capability to add and manage digital media assets in a catalog or repository. The Repository ASD provides an abstraction layer to any third-party repository service as a new standard Web service interface for digital asset management.

The following high-level Repository Abstract Service operations can be exposed by a repository or a digital asset management service provider:

- AddAsset: To create or add new media file asset to catalog or repository
- ArchiveAsset: To store or archive the asset to tape or through other archival mechanism
- RemoveAsset: To remove or delete the asset from the catalog or repository
- RetrieveAsset: To get the requested digital asset metadata information and file location from the catalog or repository
- RetrievePartialAsset: To get only certain segment of the digital asset
- SearchAsset: To search or query the catalog or repository for a list of one or more assets meeting the search criteria
- UpdateAsset: To update the media file asset in the catalog or repository
- UpdateMetaData: To update the media asset metadata in the catalog or repository
- Manage Job: To get status, cancel, stop and change priority of submitted jobs
- Manage Queue: To get current job list queued and status (length) of a job queue

These operations are shown in the WSDL in Figure B-4.

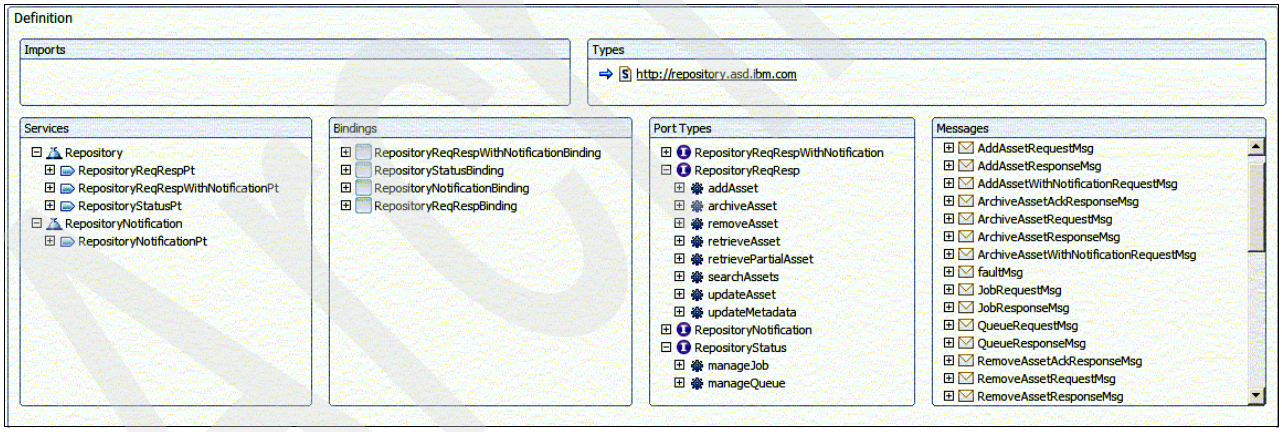

Figure B-4 Repository (DAM) Abstract Service WSDL

Media Verification Abstract Service WSDL

A media verification service provides the capability to perform automatic content verification for quality assurance process on a digital media file. The Media Verification ASD provides an abstraction layer to any third-party media content verification service as a new standard Web service interface for performing quality assurance on media files.

The following high-level Media Verification Abstract Service operations can be exposed by the service provider:

- Analyze: To perform general analysis of the digital media file
- Validate: To validate specific characteristics of the digital media file for compliance
- CreateDID: To analyze media and create an MPEG-21 DID container describing the media
- Manage Job: To get status, cancel, stop and change priority of submitted jobs
- Manage Queue: To get current job list queued and status (length) of a job queue

These operations are shown in the WSDL in Figure B-5.

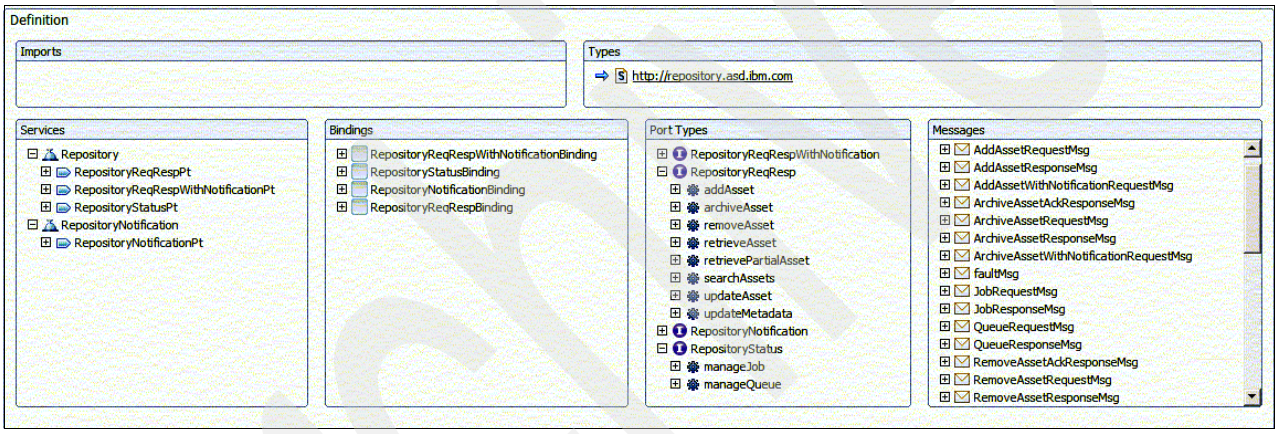

Figure B-5 Media Verification Abstract Service WSDL

C

IBM Media Hub Solution Framework

In this appendix, we introduce the IBM Media Hub Solution Framework and how an ASD is implemented in the Media Hub. The IBM Media Hub Solution Framework is a Service Oriented Architecture (SOA) based solution for the media and entertainment industry that is designed to manage the increasing complexity of running a content-focused business from the point of creation to the distribution of digital content. The architecture of the Media Hub Solution Framework is shown in Figure C-1 on page 54.

WebSphere ESB with Media Extensions

At the core of the Media Hub Solution Framework is IBM WebSphere Enterprise Service Bus (ESB) with Media Extensions (WEBS-MX or WEMX). The WEMX media extensions consist of three ESB mediation primitives, as follows:

- Service Lookup (or Endpoint Lookup)

  This primitive locates service implementations in a services registry. This mediation primitive is for finding the appropriate service based on the media formats as described by MPEG-21 specifications.

- Rule Evaluation

  This primitive is a calculation (based on MPEG-21 media metadata) of additional mediation steps required before the final service call. Actions are configured based on awareness of media formats and available transport protocols.

- Dynamic Selection

  This primitive selects among candidate service implementations, based on dynamic performance information.

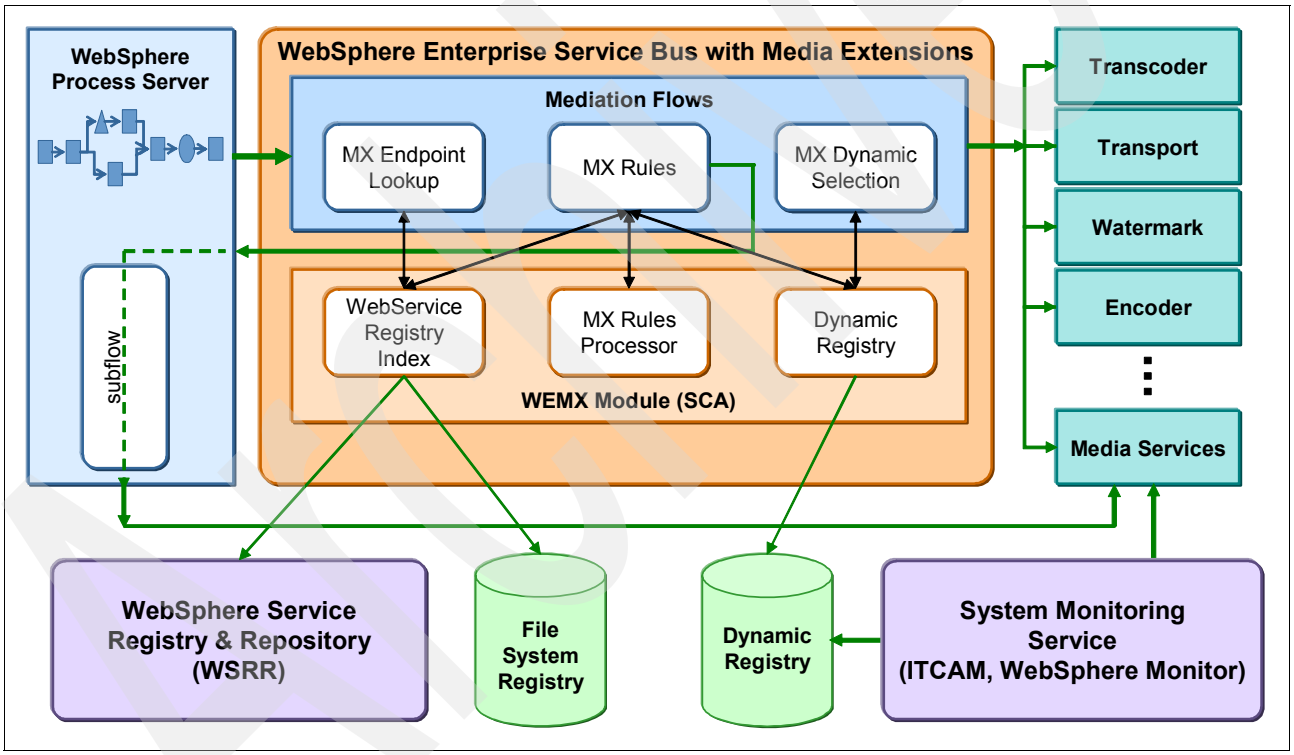

Figure C-1 IBM Media Hub Solution Framework Architecture

By leveraging the media extensions in WEMX, the Media Hub Solution Framework provides a common representation of the digital media assets across all services connected to the ESB using MPEG-21. It utilizes mediation capabilities in the WEMX extensions to intercept messages on the ESB and provides a unified and reliable mechanism to resolve media formats and location mismatches between services that create or modify media and the services that are intended to consume that media.

The WEMX extensions also provide a semantic representation of services based on media processing capabilities and an abstraction of process orchestration for supporting multiple media formats and protocols with adaptive, dynamic media flows.

Thus, within this framework, the WEMX creates the notion of *media-aware* service mediation for the Media Hub Solution. Together, the three WEMX media extensions become an essential part of the Media Hub Framework Solution to effectively orchestrate media services such as transcode, transport, or watermark in business processes.

Components of IBM Media Hub Solution Framework

Regarding the core WEMX, a number of building block components make up the Media Hub Solution Framework. The components are:

- WebSphere Service Registry and Repository (WSRR)

  WSRR is used to provide the service governance capabilities for the Media Hub Solution Framework. Working together with WEMX, WSRR allows for registering and storing media services (including the ASDs), service metadata, and endpoint information. WSRR also interacts and federates with other metadata stores that support specific phases of the media service life cycle and capture more detailed information about media services relevant in that life cycle phase. In this context, WSRR enables the Media Hub Solution Framework to manage the media services from development through deployment.

- WebSphere Business Modeler

  WebSphere Business Modeler (or Modeler) helps organizations visualize, comprehend and document their business processes, including media content operations. In the Media Hub Solution Framework, WebSphere Business Modeler is a modeling a tool for creating a visual representation of a business process that contains supporting information about the media services and operations. It is also used as a sophisticated analysis tools to evaluate both current and potential business processes and provides a direct link from business process modeling to the implementation of media services.

- Media Workflow Builder

  The Media Hub Workflow Builder is a set of tools and runtime to enable the creation, execution and monitoring of workflows related to media services. It provides a very high level of abstraction for these services and easy creation and management of the workflows. In addition, it supports changes to workflows without code change and redeployment (zero roundtrip). The tools are based on Eclipse and the runtime is a set of mediators for routing, logging, and compensation. It works in conjunction with WEMX. Media Hub Workflow Builder augments IBM industry-leading WebSphere Integration Developer by adding line-of-business (LOB) tools to IBM state-of-the-art SOA development environment.

- WebSphere Process Server (WPS)

  The WPS builds on the best-in-class WebSphere Application Server Network Deployment to provide a powerful standards-based integration platform for building and deploying composite applications within an SOA environment. WPS enables the Media Hub Solution Framework to deploy and orchestrate media integration applications and render them as services in an SOA environment. In addition to media services, WPS enables the deployment of processes in the Media Hub Solution Framework that span people (human tasks), non-media applications (invoice, billing, and so on), business rules, and the interactions among them.

- WebSphere Business Monitor

  The WebSphere Business Monitor enables you to monitor business processes in virtual real time, providing a visual display of business process status, alerts and notifications to key users, the customizable *dashboard* as WebSphere Portal pages, with scorecards, key performance indicators and gauges.

Using the IBM Media Hub Solution Framework, companies are able to transform the way they model media workflows, and manage processes through an open standards-based IT architecture that can rapidly scale up or down as market and business dynamics dictate. In addition, by using the business process management capabilities within the Media Hub Solution Framework, companies can have more control over their business challenges and find better ways to create, consolidate, administer, and distribute media content. Thus, this Framework provides a simpler, more cost-effective way to reduce the costs associated with content management, and allows media and entertainment providers to take advantage of a variety of new, multichannel distribution opportunities.

D

Additional material

This paper refers to additional material that can be downloaded from the Internet as described here.

Locating the Web material

The Web material associated with this paper is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser at:

ftp://www.redbooks.ibm.com/redbooks/REDP4464

Alternatively, you can go to the IBM Redbooks Web site at:

ibm.com/redbooks

Select **Additional Materials** and open the directory that corresponds with the IBM Redpaper form number, REDP4464.

Using the Web material

The additional Web material that accompanies this paper includes the files in Table D-1:

Table D-1 Web material files

File name	Description
7639code.zip	Zipped Java Sample Transcoder ASD adapter code
ASD_WSDLs.zip	Zipped ASD WSDL and XSD samples
ASDSampleCode.zip	Zipped .Net Sample Transcoder ASD adapter code
disclaimer.txt	Disclaimer information
SampledotNetAdapter.zip	Installation guide for .Net Sample Transcoder adapter

The Java sample Transcoder ASD adapter installation guide

The following steps describe the installation and use the sample code (see Appendix A, “Sample code for Transcoder ASD adapter” on page 41 for complete details):

1. Install Eclipse IDE for Java EE Developers (Ganymede).
2. Install Eclipse Derby plugins(derby_core_plugin_10.4.2 and derby_ui_plugin_1.1.2).
3. Install Java 6 runtime.
4. Install Apache Tomcat V6 server.
5. Add Tomcat as a server in Eclipse: **Windows®** → **Preferences** → **Server** → **Runtime Environments**, and then add Apache Tomcat V6.0 Server.
6. Unzip the sample code zip file in a folder.
7. Import the projects (ASDUtilsAxis2, data, SampleTranscoder, TranscoderAdapterAxis2) as existing projects in Eclipse.
8. Copy the file `derbyclient.jar` from the following folder:

   `ECLIPSE_ROOT\plugins\org.apache.derby.core_10.4.2\`
9. Paste it to the `TOMCAT_ROOT\lib folder.`

10. Start the Derby server:
    a. Right-click on **'data' project**.
    b. Select **Apache Derby** → **Start Derby Network server**.

       A message states that the server has started and it is ready to accept connections.
11. Create a Tomcat V6.0 server instance in Eclipse and deploy the TranscoderAxis2 project.
12. Start Tomcat server. The sample adapter is now ready to accept requests for the transcode operation.
13. Use Database Development perspective in Eclipse to view or edit the ASDSampleDB database.

    To add a database:

    a. Right-click on **Databases.**
    b. Select **New,** to create a connection profile.
    c. Give the connection profile a name (for example, ASD Database).
    d. Select **Derby**, for the profile type. Click **Next**.
    e. Change database value to ASDAdapterDB. Click **Next** and **Finish.**

    To view or edit the database table used by the sample adapter:

    a. **Select ASDAdapterDB** → **Schemas** → **ASD** → **Tables.**
    b. Right-click **Job** and then **Data** → **Edit**.

The .Net sample Transcoder ASD Adapter installation guide

This section contains a list of prerequisites, a description of the project and installable files, and instructions for testing the sample.

Prerequisites

The prerequisite software includes:

- IIS v6.0
- Microsoft® SQL Server® or SQL Express
- Microsoft .NET framework version 3.0 or later
- Visual Studio® 2008

Project and installable files

Download the SampledotNetAdapter.rar file, which contains five packages:

- AdapterASPNet_nnnn.zip

  This archive includes all of the project source files. You may open the project solution file in Visual Studio 2008.

- Config.zip

  This file includes two configuration properties files used by the Sample Transcoder: the Transcoder Adapter and the Test UI. Extract the .zip file to the `C:\temp` location. The configuration properties files are useful when you want to tune the polling interval or you want to set the service debug ports in order to intercept the HTTP traffic

- TranscoderServiceInstallFiles.zip

  This file contains the binary release files for the transcoder adapter services. Extract the file to the `wwwroot\` folder of the IIS install root. The default is the `C:\inetpub` location.

- TranscoderClientUI.zip

  This file has the Test Client as a series of Window forms. Extract it to anywhere, such as to the C:\ drive.

- DBManager.zip

  This file has the simulated reply notification service, which sends the async response back to the client callback service for async transcoder service call. It can be run as a Windows service or as a console application. In reality, the async response should leverage the async Web service support by the ASP .Net engine.

Testing the sample

The two ways to test the Sample Adapter are:

- Leverage the development server in Visual Studio 2008, which is the preferable way if you want to run the test in the debug mode to step through the code.
- Run the Web services on the real IIS server 6.0. The ASP .NET framework 3.0 or later must be installed and enabled on the IIS server.

Regardless of which way you choose to run the test, perform the following steps:

1. Create a database called ASDJOBDB on SQL Express Server. Create two tables:

   a. Create the table shown in Example D-1.

   Example: D-1 Table ASD

   ```
   CREATE TABLE ASD (GLOBALID varchar(50), LOCALID varchar(50), STATUS varchar(50),
            REPLYTO varchar(MAX), FAULTTO varchar(MAX), STARTTIME datetime,
            ENDTIME datetime, TIMEOUT, datetime, REQUESTER varchar(50),
            SOAPACTION varchar(50), WSAACTION varchar(50),
            MESSAGEID varchar(50), RELATESTO varchar(50),
            WSADRRESSVERSION varchar(50), FORMATS varchar(50),
            SPARE1 varchar(50), SPARE2 varchar(50), SPARE3 varchar(50),
            PRIMARY KEY(GLOBALID));
   ```

   b. Create the table shown in Example D-2.

   Example: D-2 Table sample Transcoder DB

   ```
   CREATE TABLE SAMPLETRANSCODERDB (JOBID varchar(50), STATUS varchar(50),
            NUMBEROFPOLLING int, PRIORITY varchar(50), OPERATION varchar(50),
            STARTTIME datetime, SOURCEMEDIA varchar(MAX),
   SPECS varchar(MAX), PRIMARY KEY(JOBID));
   ```

2. Start the DB manager:

   ```
   cd c:\DBManager
   TestDBManager
   ```

   Note: You could also install the DB manager as a Windows Service. The service installer files are in the source project files. Again, this is not necessary if you use the Async Web service framework provided by ASP .Net to implement the same functionality.

3. Start the Windows test client:

   ```
   cd C:\TranscoderClientUI
   TranscoderClient
   ```

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this paper.

Other publications

The following publication is relevant as a further information source:

- *MXF - the Material eXchange Format*, by Bruce Devlin, EBU TECHNICAL REVIEW, July 2002:

  http://www.ebu.ch/en/technical/trev/trev_291-devlin.pdf

Online resources

These Web sites are also relevant as further information sources:

- IBM Media and Entertainment

  http://www.ibm.com/media
- IBM Media Hub Solution Framework

  http://www.ibm.com/industries/media/us/detail/solution/N246120N35843T12.html
- Web Services Interoperability Organization (WS-I) Standard, Basic Profile V 1.0 and V 2.0

  http://www.ws-i.org/deliverables/workinggroup.aspx?wg=basicprofile
- Web Services Addressing 1.0 - Core W3C Recommendation

  http://www.w3.org/TR/ws-addr-core/
- Simple Object Access Protocol (SOAP) 1.1

  http://www.w3.org/TR/soap
- MPEG-21 Overview

  http://www.chiariglione.org/mpeg/standards/mpeg-21/mpeg-21.htm
- IBM developerWorks

  http://www.ibm.com/developerworks/
- Global WebSphere Community

  http://www.ibm.com/developerworks/websphere/usergroups/
  http://www.websphere.org/websphere/Site?page=home#
- MediaInfo

  http://sourceforge.net/projects/mediainfo
- Eclipse

  http://www.eclipse.org
- Apache Tomcat

  http://tomcat.apache.org

How to get Redbooks

You can search for, view, or download Redbooks, Redpapers, Technotes, draft publications and Additional materials, as well as order hardcopy Redbooks, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Abstract Service Definition for Media Services

Learn about abstract service interfaces for digital media

Implement an ASD compliant Web service

Integrate digital media in SOA enabled business processes

This IBM Redpaper publication presents the Abstract Service Definition (ASD) for digital media. The ASD provides a flexible and extensible mechanism to access classes of media services independently from the particular implementation of each service at an appropriate level of abstraction. This mechanism allows the realization of the Service Oriented Architecture (SOA) principles and simplifies the orchestration. A core value proposition of SOA is to offer a way to assemble composite applications from a set of reusable components. This approach reduces integration and maintenance costs, and allows the business to define how composite applications should be assembled to best meet the challenges it faces.

The ASD defines the different classes of media services, such as Transcoder, Encoder, Transport (for data movement), Watermark, Repository (for digital asset management), Publish, and others.). The basic concept is that an ASD interface should be focused on function and provide a standardized media service interface across different media service providers with a common terminology (ontology) to enable ease of use and reduce the system integration complexity required to implement or change a media enabled SOA deployment. This Redpaper provides guidance to implementing an ASD compliant media service interface and a framework to enable ease-of-integration in an SOA solution.

Whether you are a media service provider or an Independent Software Vendor (ISV), a Solution Integrator, a CIO, or an IT executive in an organization, this paper will prove valuable to understanding and implementing ASD media services for the enablement of digital media processes (such as content ingestion, multi-channel distribution, and so on) in a Service Oriented Architecture.

REDP-4464-00

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks