



THE MPEG-4 BOOK

Edited by Fernando Pereira and Touradj Ebrahimi

IMSC Press Multimedia Series/Andrew Tescher, Series Editor



TK
6680.5
.P47
2002

The MPEG-4 Book

Fernando Pereira
Touradj Ebrahimi
IMSC Press



Prentice Hall PTR
Upper Saddle River, NJ 07458
www.phptr.com

Library of Congress Cataloging-in-Publication Data

Pereira, Fernando C. N.

The MPEG-4 book / Fernando Pereira, Touradj Ebrahimi

p. cm.-- (IMSC Press multimedia series)

Includes bibliographical references and index.

ISBN 0-13-061621-4

1. MPEG (Video coding standard). 2. Video compression—Standards.

I. Ebrahimi, Touradj. II. Title. III. Prentice Hall IMSC multimedia series.

TK6680.5 .P47 2002

621.388—dc21

2002070433

Editorial/production supervision: *BooksCraft, Inc., Indianapolis, IN*

Acquisitions editor: *Bernard Goodwin*

Editorial assistant: *Michelle Vincenti*

Marketing manager: *Dan DePasquale*

Manufacturing manager: *Alexis Heydt-Long*

Cover design director: *Jerry Votta*

Cover designer: *Anthony Gemmellaro*

Full-service production manager: *Anne Garcia*



© 2002 Pearson Education, Inc.

Publishing as Prentice Hall PTR

Upper Saddle River, New Jersey 07458

Prentice Hall books are widely used by corporations and government agencies for training, marketing, and resale.

For information regarding corporate and government bulk discounts please contact:
Corporate and Government Sales (800) 382-3419 or corpsales@pearsontechgroup.com

Company and product names mentioned herein are the trademarks or registered trademarks of their respective owners.

All rights reserved. No part of this book may be reproduced, in any form or by any means, without permission in writing from the publisher.

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

ISBN 0-13-061621-4

Pearson Education LTD.

Pearson Education Australia PTY, Limited

Pearson Education Singapore, Pte. Ltd.

Pearson Education North Asia Ltd.

Pearson Education Canada, Ltd.

Pearson Educación de México, S.A. de C.V.

Pearson Education—Japan

Pearson Education Malaysia, Pte. Ltd.

Contents

Foreword	xix
Preface	xxiii
Abbreviations	xxvii
1 Context, Objectives, and Process	1
1.1 MPEG-4 Objectives	4
1.1.1 Functionalities	5
1.1.2 Requirements	7
1.1.3 Tools	10
1.1.4 Applications	12
1.2 Formal Standardization Process	13
1.3 MPEG Modus Operandi	15
1.3.1 Mission	16
1.3.2 Principles	17
1.3.3 Standards Development Approach	18
1.4 MPEG-4 Standard Organization	21
1.5 MPEG-4 Schedule	25
1.6 MPEG-4 Industry Forum	30
1.7 Summary	31
1.8 References	32

2 MPEG-4 Overview	37
2.1 Design Goals.....	38
2.2 An End-to-End Walkthrough.....	40
2.3 Terminal Architecture.....	43
2.4 MPEG-4 Tools.....	47
2.4.1 Systems Tools	47
2.4.2 Visual Tools.....	48
2.4.3 Audio Tools.....	50
2.4.4 DMIF Tools	51
2.4.5 Other MPEG-4 Tools	51
2.4.6 Profiles and Levels	52
2.5 MPEG-4 and Other Multimedia Standards	52
2.6 MPEG-4 Applications	56
2.6.1 Multimedia Portals.....	56
2.6.2 Interactive Broadcasting.....	57
2.6.3 Multimedia Conferencing and Communities	59
2.7 Summary	60
2.8 References	61
3 Object Description and Synchronization	65
3.1 Object Descriptors: Entry Points to MPEG-4 Content.....	65
3.1.1 Syntactic Description Language	66
3.1.2 Object Description	70
3.1.3 Stream Description.....	72
3.1.4 Stream Relationship Description.....	76
3.1.5 Content Complexity Description.....	77
3.1.6 Streaming ODs	79
3.1.7 Linking a Scene to Its Media Streams	82
3.1.8 MPEG-4 Content Access Procedure.....	85
3.2 Semantic Description and Access Management.....	86
3.2.1 Object Content Information: Meta Information About Objects	86
3.2.2 Intellectual Property Management and Protection	89
3.3 Timing Model and Synchronization of Streams	90
3.3.1 Modeling Time	90
3.3.2 Time Stamps and Access Units.....	92
3.3.3 Packetizing Streams: The Sync Layer.....	93
3.3.4 Distributed Content, Time Bases, and OCR Streams	96
3.3.5 Media Time	98
3.3.6 System Decoder Model	98
3.4 Summary	100
3.5 References	101
4 BIFS: Scene Description	103
4.1 Basics of BIFS	104
4.1.1 Scene and Nodes	104
4.1.2 Fields and ROUTEs.....	106

4.1.3 Node Types	108
4.1.4 Subscenes and Hyperlinks	109
4.1.5 Scene Changes	109
4.1.6 Scene Rendering	112
4.1.7 Binary Encoding	113
4.1.8 Quantization	114
4.2 Basic BIFS Features by Example	114
4.2.1 Trivial Scene	114
4.2.2 Movie with Subtitles	117
4.2.3 Icons and Buttons	119
4.2.4 Slides and Transitions	121
4.2.5 Simple 3D Scene	124
4.2.6 Magnifying Glass	128
4.3 Advanced BIFS Features	129
4.3.1 Scripting	130
4.3.2 Encapsulation and Reuse	131
4.3.3 Text Layout	134
4.4 A Peek Ahead on BIFS	136
4.4.1 Media Nodes	136
4.4.2 New Sensors	137
4.4.3 FlexTime	138
4.4.4 ServerCommand	139
4.5 Profiles	139
4.6 All BIFS Nodes	140
4.7 Summary	146
4.8 References	147
5 MPEG-J: MPEG-4 and Java	149
5.1 MPEG-J Architecture	154
5.1.1 MPEGlets	156
5.1.2 Delivery	158
5.1.3 Security	159
5.2 MPEG-J APIs	160
5.2.1 Terminal APIs	163
5.2.2 Scene APIs	164
5.2.3 Resource APIs	173
5.2.4 Decoder APIs	176
5.2.5 Network APIs	178
5.2.6 Service Information and Section Filtering APIs	180
5.2.7 MPEG-J Profiles	180
5.3 Application Scenarios	181
5.3.1 Adaptive Rich Media Content for Wireless Devices	181
5.3.2 Enhanced Interactive Electronic Program Guide (EPG)	181
5.3.3 Enriched Interactive Digital Television	182
5.3.4 Content Personalization	182

5.4 Reference Software	182
5.5 Summary	183
5.6 References	185
6 Extensible MPEG-4 Textual Format.....	187
6.1 Objectives	188
6.2 Cross-Standard Interoperability	189
6.3 XMT Two-Tier Architecture	190
6.4 XMT-Ω Format	193
6.4.1 Reusing SMIL in XMT-Ω.....	194
6.4.2 Extensible Media (xMedia) Objects	195
6.4.3 Timing and Synchronization.....	198
6.4.4 Time Manipulations	202
6.4.5 Animation.....	203
6.4.6 Spatial Layout	205
6.4.7 XMT-Ω Examples.....	205
6.5 XMT-A Format.....	210
6.5.1 Document Structure	210
6.5.2 Timing	211
6.5.3 Scene Description	212
6.5.4 Object Descriptor Framework.....	219
6.5.5 Deterministic Mapping of XMT-A	222
6.5.6 Interoperability with X3D	223
6.6 Summary	224
6.7 References	224
7 Transporting and Storing MPEG-4 Content	227
7.1 Delivery Framework	229
7.1.1 DMIF-Application Interface.....	231
7.1.2 DMIF Network Interface.....	238
7.1.3 Existent Signaling Protocols	246
7.2 FlexMux Tool	248
7.2.1 Timing of a FlexMux Stream	251
7.3 MPEG-4 File Format.....	253
7.3.1 Temporal Structure: Tracks and Streams, Time and Durations	254
7.3.2 Physical Structure: Atoms and Containers, Offsets and Pointers.....	257
7.3.3 MPEG-4 Systems Concepts in MP4.....	259
7.3.4 MPEG-4 Track Types and Storage	259
7.3.5 Hinting	260
7.3.6 Atoms.....	262
7.3.7 Random Access	265
7.3.8 An MP4 Example.....	266
7.3.9 Summary of MP4	268
7.4 Transporting MPEG-4 over MPEG-2.....	268
7.4.1 Brief Introduction to MPEG-2 Systems.....	269
7.4.2 Transport of MPEG-4 Elementary Streams over MPEG-2 Systems	274

7.4.3 Transport of MPEG-4 Scenes over MPEG-2 Systems	275
7.5 Transporting MPEG-4 over IP	280
7.5.1 Brief Introduction to Streaming over IP	281
7.5.2 Transport of Elementary Streams over IP	283
7.5.3 Transport of SL-Packetized Streams over IP	285
7.5.4 FlexMux Streams over IP and Timing Models	288
7.6 Summary	289
7.7 References	289
8 Natural Video Coding.....	293
8.1 General Overview	294
8.1.1 Functionalities and Application Scenarios	294
8.1.2 Basic Principles	295
8.2 Coding of Rectangular Video Objects.....	300
8.2.1 Overview	300
8.2.2 New Motion-Compensation Tools.....	303
8.2.3 New Texture Coding Tools.....	311
8.3 Coding of Arbitrarily Shaped Video Objects.....	318
8.3.1 Binary Shape Coding	318
8.3.2 Gray-Level Shape Coding	323
8.3.3 Coding of Boundary Macroblocks	324
8.4 Scalable Video Coding	330
8.4.1 Spatial Scalability	332
8.4.2 Temporal Scalability	337
8.4.3 SNR Fine Granularity Scalability	340
8.5 Special Video Coding Tools.....	343
8.5.1 Interlaced Coding	344
8.5.2 Error-Resilient Coding	345
8.5.3 Reduced Resolution Coding	355
8.5.4 Sprite Coding	356
8.5.5 Short Video Header Mode	359
8.5.6 Texture Coding for High-Quality Applications.....	359
8.6 Visual Texture Coding	361
8.6.1 VTC Tools	363
8.6.2 Wavelet Coding.....	364
8.6.3 Shape-Adaptive Wavelet Coding	368
8.6.4 Spatial and Quality Scalability	369
8.6.5 Bitstream Packetization	371
8.6.6 Tiling	374
8.7 Summary	375
8.8 References	376
9 Visual SNHC Tools	383
9.1 SNHC Overview.....	384
9.1.1 VRML, X3D: Why Is SNHC Needed?.....	385
9.1.2 SNHC Visual Tools.....	388

9.2 Face and Body Animation	389
9.2.1 Overview	389
9.2.2 Default Facial Expression and Body Posture	392
9.2.3 FBA Object	393
9.2.4 FDP and BDP Listing and Coding	396
9.2.5 FAP and BAP Listing and Coding	403
9.2.6 FBA and Text-to-Speech Interface	410
9.3 2D Mesh Coding	411
9.3.1 2D Mesh Object	412
9.3.2 Coding Scheme	413
9.3.3 Example	420
9.4 3D Mesh Coding	421
9.4.1 3D Mesh Object	422
9.4.2 Coding Scheme	425
9.4.3 Examples	437
9.5 View-Dependent Scalability	440
9.5.1 View-Dependent Object	442
9.5.2 Coding Scheme	444
9.5.3 Example	446
9.6 Profiles and Levels	446
9.7 Summary	447
9.8 Acknowledgments	447
9.9 References	448
10 Speech Coding	451
10.1 Introduction to Speech Coding	452
10.2 Overview of MPEG-4 Speech Coders	454
10.3 MPEG-4 CELP Coding	455
10.3.1 CELP Encoder	455
10.3.2 CELP Decoder	456
10.3.3 Parameter Decoding	457
10.3.4 Multipulse Excitation	458
10.3.5 Regular Pulse Excitation	459
10.3.6 Scalability	460
10.3.7 Silence Compression	461
10.4 MPEG-4 HVXC Coding	461
10.4.1 HVXC Encoder	462
10.4.2 HVXC Decoder	470
10.4.3 Variable Bit-Rate Coding	474
10.5 Error Robustness	475
10.5.1 Error-Resilient HVXC	477
10.5.2 Error-Resilient CELP	481
10.6 Summary	482
10.7 References	483

11 General Audio Coding.....	487
11.1 Introduction to Time/Frequency Audio Coding	488
11.2 MPEG-2 Advanced Audio Coding	490
11.2.1 Coder Overview	491
11.2.2 Gain Control	491
11.2.3 Filterbank	493
11.2.4 Quantization	495
11.2.5 Noiseless Coding.....	497
11.2.6 Temporal Noise Shaping.....	499
11.2.7 Prediction.....	502
11.2.8 Joint Stereo Coding	504
11.2.9 Bitstream Multiplexing.....	507
11.2.10 Other Aspects	508
11.3 MPEG-4 Additions to AAC	508
11.3.1 Perceptual Noise Substitution.....	509
11.3.2 Long-Term Prediction	511
11.3.3 TwinVQ.....	513
11.3.4 Low-Delay AAC (AAC-LD).....	515
11.3.5 Error Robustness.....	517
11.4 MPEG-4 Scalable Audio Coding	518
11.4.1 Large-Step Scalable Audio Coding	519
11.4.2 Bit-Sliced Arithmetic Coding.....	523
11.5 Introduction to Parametric Audio Coding	525
11.5.1 Source and Perceptual Models.....	526
11.5.2 Parametric Encoding and Decoding Concepts.....	528
11.6 MPEG-4 HILN Parametric Audio Coding	530
11.6.1 HILN Parametric Audio Encoder	530
11.6.2 HILN Bitstream Format and Parameter Coding	535
11.6.3 HILN Parametric Audio Decoder	538
11.7 Summary	539
11.8 Acknowledgments	540
11.9 References	540
12 SNHC Audio and Audio Composition.....	545
12.1 Synthetic-Natural Hybrid Coding of Audio.....	546
12.2 Structured Audio Coding.....	548
12.2.1 Algorithmic Synthesis and Processing.....	549
12.2.2 Wavetable Synthesis	552
12.2.3 Interface Between Structured Audio Coding and AudioBIFS	554
12.2.4 Structured Audio Applications	554
12.3 Text-to-Speech Interface.....	555
12.4 Audio Composition.....	557
12.4.1 VRML Sound Model in BIFS	558
12.4.2 Other AudioBIFS Nodes	560
12.4.3 Enhanced Modeling of 3D Audio Scenes in MPEG-4	567

12.4.4 Advanced AudioBIFS for Enhanced Presentation of 3D Sound Scenes.....	569
12.5 Summary	580
12.6 References	580
13 Profiling and Conformance: Approach and Overview.....	583
13.1 Profiling and Conformance: Goals and Principles.....	584
13.2 Profiling Policy and Version Management.....	588
13.3 Overview of Profiles in MPEG-4	592
13.3.1 Visual Profiling.....	592
13.3.2 Audio Profiling.....	602
13.3.3 Graphics Profiling.....	609
13.3.4 Scene Graph Profiling	613
13.3.5 Object Descriptor Profiling.....	619
13.3.6 MPEG-J Profiling	619
13.4 Summary	620
13.5 Acknowledgements.....	620
13.6 References	620
14 Implementing the Standard: The Reference Software.....	623
14.1 Reference Software Modules	625
14.2 Systems Reference Software	625
14.3 MPEG-4 Player Architecture.....	628
14.3.1 Player Structure	631
14.4 Scene Graph	632
14.4.1 Parsing the Scene Graph.....	633
14.4.2 Rendering the Scene Graph	638
14.5 PROTOs	639
14.6 Synchronization	643
14.6.1 Sync Layer	643
14.6.2 MediaStream Objects.....	644
14.7 Object Descriptors.....	646
14.7.1 Syntactic Description Language	646
14.7.2 Parsing the OD Stream	647
14.7.3 Execution of OD Objects.....	651
14.8 Plug-Ins	652
14.8.1 DMIF Plug-Ins.....	652
14.8.2 Decoder Plug-Ins.....	654
14.8.3 IPMP Plug-Ins	656
14.9 2D Compositor.....	656
14.9.1 Using the Core Framework.....	657
14.9.2 DEF and USE Handling.....	657
14.9.3 Rendering Optimization.....	658
14.9.4 Synchronization	659
14.10 3D Compositor.....	660
14.10.1 Differences Between the 2D and 3D Compositors	661

14.10.2 Using the Core Framework.....	662
14.10.3 Overview of the Key Classes.....	662
14.10.4 3D Rendering Process	662
14.10.5 Support for 2D Nodes	664
14.10.6 User Navigation	664
14.11 Summary.....	666
14.12 References	666
15 Video Testing for Validation	669
15.1 General Aspects	670
15.1.1 Selection of Test Material	671
15.1.2 Selection of Test Subjects.....	671
15.1.3 Laboratory Setup.....	672
15.1.4 Test Plan	672
15.1.5 Training Phase	673
15.2 Test Methods	673
15.2.1 Single Stimulus Method.....	674
15.2.2 Double Stimulus Impairment Scale Method	675
15.2.3 Double Stimulus Continuous Quality Scale Method.....	676
15.2.4 Simultaneous Double Stimulus for Continuous Evaluation Method ...	677
15.3 Error-Resilience Test	680
15.3.1 Test Conditions.....	681
15.3.2 Test Material	684
15.3.3 Test Method and Design	684
15.3.4 Data Analysis	686
15.3.5 Test Results	686
15.4 Content-Based Coding Test.....	688
15.4.1 Test Conditions.....	688
15.4.2 Test Material	689
15.4.3 Test Method and Design	689
15.4.4 Data Analysis	690
15.4.5 Test Results	690
15.5 Coding Efficiency for Low and Medium Bit-Rate Test.....	691
15.5.1 Test Conditions.....	691
15.5.2 Test Material	692
15.5.3 Test Method and Design	692
15.5.4 Data Analysis	693
15.5.5 Test Results	693
15.6 Advanced Real-Time Simple Profile Test.....	694
15.6.1 Test Conditions.....	696
15.6.2 Test Material	699
15.6.3 Test Method and Design	700
15.6.4 Data Analysis	701
15.6.5 Test Results	702
15.7 Summary	704

15.8 References	705
16 Audio Testing for Validation.....	709
16.1 General Aspects.....	710
16.1.1 Selection of Test Material	710
16.1.2 Selection of Test Subjects.....	711
16.1.3 Laboratory Setup	712
16.1.4 Test Plan	712
16.1.5 Training Phase.....	712
16.2 Test Methods	713
16.2.1 Absolute Category Rating Method.....	713
16.2.2 Paired Comparison Method	714
16.2.3 MUSHRA Method.....	715
16.3 Narrowband Digital Audio Broadcasting Test	717
16.3.1 Test Conditions	718
16.3.2 Test Material	719
16.3.3 Assessment Method and Test Design.....	720
16.3.4 Data Analysis.....	721
16.3.5 Test Results	722
16.4 Audio on the Internet Test.....	724
16.4.1 Test Conditions	724
16.4.2 Test Material	725
16.4.3 Assessment Method and Test Design.....	726
16.4.4 Data Analysis.....	727
16.4.5 Test Results	728
16.5 Speech Communication Test	730
16.5.1 Test Conditions	730
16.5.2 Test Material	732
16.5.3 Assessment Method and Test Design.....	733
16.5.4 Data Analysis.....	734
16.5.5 Test Results	734
16.6 Version 2 Coding Efficiency Test	736
16.6.1 Test Conditions	737
16.6.2 Test Material	739
16.6.3 Assessment Method and Test Design.....	740
16.6.4 Data Analysis.....	740
16.6.5 Test Results	740
16.7 Version 2 Error-Robustness Test.....	744
16.7.1 Test Conditions	744
16.7.2 Test Material	747
16.7.3 Test Method and Experimental Design.....	747
16.7.4 Data Analysis.....	747
16.7.5 Test Results	747
16.8 Summary	749

16.9 References	750
A Levels for Visual Profiles.....	753
A.1 Video Buffering Verifier Mechanism	754
A.1.1 Video Rate Buffer Verifier Definition	756
A.1.2 Video Complexity Verifier Definition	760
A.1.3 Video Reference Memory Verifier Definition	764
A.1.4 Interaction Between the VBV, VCV, and VMV Models.....	766
A.2 Definition of Levels for Video Profiles	767
A.3 Definition of Levels for Synthetic Profiles	767
A.3.1 Scalable Texture Profile	775
A.3.2 Simple Face Animation Profile	775
A.3.3 Simple FBA Profile.....	775
A.3.4 Advanced Core and Advanced Scalable Texture Profiles	776
A.4 Definition of Levels for Synthetic and Natural Hybrid Profiles	776
A.4.1 Basic Animated Texture Profile	776
A.4.2 Hybrid Profile.....	778
A.5 References.....	778
B Levels for Audio Profiles	781
B.1 Complexity Units	781
B.2 Definition of Levels for Audio Profiles.....	783
B.2.1 Main Profile	784
B.2.2 Scalable Profile.....	784
B.2.3 Speech Profile.....	785
B.2.4 Synthetic Profile.....	785
B.2.5 High-Quality Audio Profile	786
B.2.6 Low-delay Audio Profile	786
B.2.7 Natural Audio Profile	786
B.2.8 Mobile Audio Internetworking Profile.....	786
B.3 References.....	788
C Levels for Graphics Profiles	789
C.1 Simple 2D Profile	789
C.2 Simple 2D + Text Profile.....	790
C.3 Core 2D Profile	790
C.4 Advanced 2D Profile	793
C.5 References.....	795
D Levels for Scene Graph Profiles	797
D.1 Simple 2D Profile	797
D.2 Audio Profile.....	798
D.3 3D Audio Profile	801
D.4 Basic 2D Profile	802
D.5 Core 2D Profile	802

D.6 Advanced 2D Profile.....	806
D.7 Main 2D Profile	809
D.8 References.....	813
E MPEG-J Code Samples.....	815
E.1 Scene APIs	815
E.2 Resource and Decoder APIs	817
E.2.1 Listener Class for Decoder Events.....	819
E.2.2 Listener Class for Renderer Events.....	820
E.3 Network APIs	820
E.4 Section Filtering APIs	821
Index.....	823

Context, Objectives, and Process¹

by *Fernando Pereira*

Keywords: audiovisual object-based representation, audiovisual objects, MPEG-4, standardization process

“What does it mean, to see? The plain man’s answer (and Aristotle’s, too) would be, to know what is where by looking. In other words, vision is the process of discovering from images what is present in the world, and where it is.” [Marr82]

The first video coding standards and their underlying representation models mainly address the vision process by providing video representation in the form of a sequence of rectangular 2D frames, giving users a window to the real world: the television paradigm. However, the process of vision is often just the initial part of the task at hand, because typically humans need and want to see, to take actions after, to interact with the objects identified. A similar reasoning can be made regarding the process of hearing and the corresponding audio representation models [Pere99].

1. Some sections of this chapter are adapted from *Signal Processing: Image Communication*, 15 (4–5) (2000), Fernando Pereira, pp. 271–279, “MPEG-4: Why, What, How and When?”, Copyright (2000), with permission from Elsevier Science.

Although the television paradigm dominated audiovisual communications for many years, the situation has been evolving quickly in terms of the ways audiovisual content is produced, delivered, and consumed [KoPC97]. Moreover, hardware and software are becoming increasingly powerful, with microelectronic technology providing new programmable processors, opening new frontiers to the representation technologies used and to the functionalities provided.

Producing content today is easier than ever before. Digital still cameras directly storing in JPEG format have hit the mass market. Together with the first digital video cameras directly recording in MPEG-1 format, this represents a major step for consumer acceptance of digital audiovisual acquisition and compression technology. This step transforms every individual into a potential content producer, capable of creating content that can be easily distributed and published on the Internet. In addition, more content is being synthetically produced—that is, computer generated—and integrated with natural material in truly hybrid audiovisual content. The various pieces of content, digitally encoded, can be successively reused without the quality losses typical of the previous analog processes.

Whereas audiovisual information, notably the visual portion, until recently was carried only over very few networks, the trend is now toward the generalization of visual information in every single network. Moreover, the increasing mobility in telecommunications is a major trend. Mobile connections will not be limited to voice; other types of data, including real-time media, are already emerging. Because mobile telephones are replaced every two to three years, new mobile devices can finally make the decade-long promise of audiovisual communications a reality. Notice that the need for visual communication is much stronger when people are not at home, and so have something to show besides the usual living room—for example, the nice beach where they are vacationing. This reinforces the relevance of audiovisual mobile communications.

The explosion of the World Wide Web and the acceptance of its interactive mode of operation have clearly shown that the traditional television paradigm will no longer suffice for audiovisual services. Users want to have access to audio and video as they now have access to text and graphics. This requires moving pictures and audio of acceptable quality at low bit rates on the Web, and Web-type interactivity with live content. It will be possible to activate relationships between users (in a potentially virtual world) through hyperlinking—the Web paradigm—and to experience interactive immersion in natural and virtual environments—the games paradigm.

As many of the emerging audiovisual applications demanded interworking, the need to develop an open and timely international standard addressing the needs mentioned above became evident. In 1993, MPEG (Moving Picture Experts Group) [MPEG] launched the MPEG-4 project, later formally called *Coding of Audio-Visual Objects*, to address (among other things) the requirements associated with the new applications resulting from these trends [N1177, N4319, N4505].

The need for any standard comes from an essential requirement relevant for all applications involving communication between two or more parts: *interoperability*. Interoperability is thus the requirement expressing the user's dream of exchanging any type of information without any technical barriers, in the simplest way. Without a standard way to perform some of the operations involved in the communication process and to structure the data exchanged, easy interoperability between terminals would be impossible. Having said that, it is clear that a standard shall specify the minimum number of tools needed to guarantee interoperability (because it is important that as many as possible non-normative technical zones exist), to allow the incorporation of technical advances, and thus to increase the lifetime of the standard, as well as to stimulate industrial technical competition. The existence of a standard also has important economic implications, because it allows the sharing of costs and investments and the acceleration of applications deployment.

MPEG has been responsible for the successful MPEG-1 (ISO/IEC 11172) and MPEG-2 (ISO/IEC 13818) standards, which have given rise to widely adopted commercial products and services, such as Video-CD, DVD, digital television, digital audio broadcasting (DAB), and MP3 (MPEG-1 Audio layer 3) players and recorders. The MPEG-4 standard (ISO/IEC 14496) is aimed at defining an audiovisual coding standard to address the emerging needs of the communication, interactive, and broadcasting service models as well as the needs of the mixed service models resulting from their convergence. The apparent convergence of the three traditionally separate application areas—communications, computing, and TV/film/entertainment—was evident in their cross-fertilization, with functionalities characteristic of each area increasingly emerging in the others (e.g., personal communications including video information or entertainment including interactive capabilities).

Following the previous successes—in fact, as a natural consequence of the vision underpinning MPEG-4—MPEG initiated in 1996 another standardization project addressing the problem of describing audiovisual content to allow the quick and efficient searching, processing, and filtering of various types of multimedia material: MPEG-7 (ISO/IEC 15938), officially called *Multimedia Content Description Interface* [N4509]. In fact, digital audiovisual information is more and more accessible to everyone, not only in terms of consumption but also in terms of production. But if it is much easier today to acquire, process, and distribute audiovisual content, it must be equally easy to access the available information, because huge amounts of audiovisual information are being generated all over the world every day. The need for a powerful way to quickly and efficiently identify, search, and filter various types of audiovisual content, by humans or machines (also using non-text-based technologies), directly follows from the urge to efficiently use the available audiovisual content and the difficulty of doing so. MPEG-7 will specify a standard way of describing various types of audiovisual information, including still pictures, video, speech, audio, graphics, 3D models, and synthetic audio, regardless of their representation format (e.g., analog or digital) and storage support (e.g.,

paper, film, or tape). In comparison with other available or emerging solutions for audiovisual content description, MPEG-7 can be mainly distinguished by (a) being general purpose, meaning its ability to describe content from many application environments; (b) its object-based representation model, meaning the capability of independently describing individual objects within a scene (be it MPEG-4 or any other format); (c) the integration of low-level and high-level features/descriptors into a single description framework, allowing it to combine the power of both types of descriptors; and (d) its extensibility, provided by the Description Definition Language, which allows MPEG-7 to keep growing, to be extended to new application areas, to answer newly emerging needs, and to integrate novel description tools [PeKo99]. The MPEG-7 standard was finalized in the summer of 2001.

Following the development of standards addressing more focused targets, MPEG acknowledged the lack of a big picture that described how the various elements building the infrastructure for the deployment of applications using multimedia content relate to each other, or even if there are missing standard specifications for some of these elements [N4333]. To address this problem, MPEG started the MPEG-21 project (first ISO/IEC 18034, now ISO/IEC 21000), formally called *Multimedia Framework*, with the aim of understanding if and how these various elements fit together, and to discuss which new standards might be required if gaps in the infrastructure exist. Once this work has been carried out, new standards will be developed for the missing elements with the involvement of other bodies, where appropriate; finally, the existing and novel standards will be integrated in the MPEG-21 multimedia framework. The MPEG-21 vision is thus to define a multimedia framework to enable transparent and augmented use of multimedia resources across a wide range of networks and devices used by different communities [N4333]. The MPEG-21 multimedia framework will identify and define the key elements needed to support the multimedia value and delivery chain, as well as the relationships between and the operations supported by them [N4511].

After briefly covering the context that motivated the birth of the MPEG-4 project, this chapter presents its major objectives in terms of functionalities, requirements, tools, and applications, as well as its organization and the sequence followed to achieve the goals defined. This chapter also addresses the MPEG modus operandi, notably its mission, principles, and specific approach to the development of standards. Finally, the objectives and working approach of the MPEG-4 Industry Forum will be presented.

1.1 MPEG-4 OBJECTIVES

Although MPEG discussions about projects beyond MPEG-2 began as early as May 1991, at the Paris MPEG meeting, it was not until September 1993 that the MPEG Applications and Operational Environments (AOE) subgroup was set up and met for the first time. The main task of this subgroup was to identify

the applications and requirements relevant to the far-term, very low bit-rate coding solution to be developed by International Organization for Standardization (ISO)/MPEG as stated in the initial MPEG-4 project description [N271]. At the same time, the near-term hybrid coding solution being developed within the International Telecommunications Union-Telecommunications Standardization Sector (ITU-T) Low Bit-rate Coding (LBC) group started producing the first results (later, the ITU-T H.263 standard [H263]). It was then generally felt that those results were close to the best performance that could be obtained by block-based, hybrid, DCT/motion-compensation video coding schemes.

In July 1994, the Grimstadt MPEG meeting marked a major change in the direction of MPEG-4. Until that meeting, the main goal of MPEG-4 was to obtain a significantly better compression ratio than could be achieved by conventional coding techniques. Few people, however, believed it was possible, in the next five years, to make enough improvements over the LBC standard (H.263 and H.263+) to justify a new standard.² So the AOE subgroup was faced with the need to broaden the objectives of MPEG-4, believing that pure compression gains would not be enough to start a new MPEG standardization project. The subgroup then began an in-depth analysis of the audiovisual world trends, based on the convergence of the TV/film/entertainment, computing, and telecommunications worlds. The conclusion was that the emerging MPEG-4 coding standard should support new ways (notably content-based) of communicating, accessing, and manipulating digital audiovisual data.

1.1.1 Functionalities

Following this change of direction and the analysis made, the vision driving the MPEG-4 standard was explained through the eight new or improved functionalities described in the MPEG-4 Proposal Package Description (PPD), prepared by the time of the first MPEG-4 call for proposals in July 1995 [N998]. These eight functionalities came from an assessment of the functionalities that would be useful in future audiovisual applications, but which were not supported (or at least not well supported) by the available coding standards. The eight new or improved functionalities were clustered in three classes related to the three worlds—TV/film/entertainment, computing, and telecommunications—the convergence of which MPEG-4 wanted to address [N998, PeKo96]:

1. Content-based interactivity

- ✎ **Content-based multimedia data access tools:** MPEG-4 shall provide efficient data access and organization based on the audiovisual content. Access tools may be indexing, hyperlinking, querying, browsing, uploading, downloading, and deleting. Sample uses include content-based

2. At least under the acceptable complexity boundaries.

retrieval of information from online libraries and travel information databases.³

- ✎ **Content-based manipulation and bitstream editing:** MPEG-4 shall provide syntax and coding schemes to support content-based manipulation and bitstream editing without the need for transcoding. This means the user should be able to select one specific object in the scene/bitstream and change some of its characteristics. Sample uses include home movie production and editing, interactive home shopping, and the insertion of a sign language interpreter or subtitles.
- ✎ **Hybrid natural and synthetic data coding:** MPEG-4 shall support efficient methods for combining synthetic scenes with natural scenes (e.g., text and graphics overlays), the ability to code and manipulate natural and synthetic audio and visual data, and decoder-controllable methods of mixing synthetic data with ordinary video and audio (allowing for interactivity). For example, in virtual reality applications, animations and synthetic audio (e.g., MIDI) can be mixed with ordinary audio and video in games, and graphics can be rendered from different viewpoints.
- ✎ **Improved temporal random access:** MPEG-4 shall provide efficient methods to randomly access, within a limited time and with fine resolution, parts (e.g., frames or objects) from an audiovisual sequence. Example usage: audiovisual data can be randomly accessed from a remote terminal over limited-capacity media, a fast-forward can be performed on a single audiovisual object in the sequence.

2. Compression efficiency

- ✎ **Improved coding efficiency:** The growth of mobile networks creates an ongoing demand for improved coding efficiency; therefore, MPEG-4 set as its target providing subjectively better audiovisual quality compared to existing or other emerging standards (such as H.263), at comparable bit rates. Sample uses include efficient transmission of audiovisual data on low-bandwidth channels and efficient storage of audiovisual data on limited-capacity media, such as chip cards.
- ✎ **Coding of multiple concurrent data streams:** MPEG-4 shall provide the ability to efficiently code multiple views/soundtracks of a scene as well as sufficient synchronization between the resulting elementary streams.⁴ For stereoscopic and multiview video applications, MPEG-4 shall include the ability to exploit redundancy in multiple views of the

3. Although this is one of the eight initial MPEG-4 functionalities, it relates more to the tools developed later in the context of the MPEG-7 standard [N4509].

4. This seems to be the only initial MPEG-4 functionality that has not been addressed at the end of 2001, as the companies participating in MPEG have shown no significant interest in it.

same scene, also permitting solutions that allow compatibility with normal video. Sample uses include multimedia entertainment (e.g., virtual reality games and 3D movies), training and flight simulations, multimedia presentations, and education.

3. Universal access

- ☞ **Robustness in error-prone environments:** Because universal accessibility implies access to applications over many wireless and wired networks and storage media, MPEG-4 shall provide an error robustness capability. Particularly, sufficient error robustness shall be provided for low bit-rate applications under severe error conditions. Sample uses include transmission from a database over a wireless network, communicating with a mobile terminal, and gathering audiovisual data from a remote location.
- ☞ **Content-based scalability:** MPEG-4 shall provide the ability to achieve scalability with a fine granularity in content, spatial resolution, temporal resolution, quality, and complexity. Content-scalability may imply the existence of a prioritization of the objects in the scene. Sample uses include user selection of decoded quality of individual objects in the scene and database browsing at different scales, resolutions, and qualities.

These functionalities were essential to shaping the MPEG-4 vision, balancing completely new functionalities with more traditional ones, and thus allowing a bridge from the past to the future not only in terms of functionalities but also in terms of tools and experts.

1.1.2 Requirements

Following the identification of the fundamental MPEG-4 functionalities, MPEG started a requirements development process, which has been continuously evolving since then. The requirements serve to drive the tools development process and assure that the right technology is being specified: Tools will be developed to fulfill the identified requirements, and no tools that do not address any requirement will be defined.

By the middle of 2001, the MPEG-4 requirements were structured as shown in Table 1.1. The requirements are organized in terms of major technical areas, which do not directly correspond either to Parts of the MPEG-4 standard or to MPEG working subgroups. Table 1.1 gives only a general flavor of the requirements (for the details, see [N4319]).

Within each category, the requirements are to be fulfilled by the set of tools selected for the standard, and not all requirements must be addressed by each individual tool. It is the right combination of tools that allows building the algorithms that can address the specific needs of a certain class of applications.

Table 1.1 MPEG-4 requirements [N4319]

Requirements for systems	
Flexibility	Multipoint operation
Multiplexing of audio, visual, and other information	Object content information (OCI)
Composition of audio and visual objects	Video-related metadata
Application texture	Delay
Downloading	Configuration modes
User interaction	Priority of audiovisual objects
Media interworking	Dynamic resource management
Compatibility	Reference to associated MPEG-7 data
Robustness to information errors and losses	File format
Object-based bitstream manipulation and editing	Textual format
Requirements for natural video objects	
Object-based representation	Object-based coding flexibility
Video content	Object-based scalability
Object-based bitstream manipulation and editing	Delay modes
Object-based random access	Formats
Object quality and fidelity	Bit-rate modes
Coding of multiple concurrent data streams	Complexity modes
Robustness to information errors and losses	Still images
	Tandem coding*
Requirements for synthetic video objects	
Types of synthetic video objects	Text overlay
2D/3D mesh compression	Image and graphics overlay
Definition and animation parameter compression	View-dependent texture scalability
Texture mapping	Geometrical transformations
	Video object tracking
Requirements for natural audio objects	
Object-based representation	Robustness to information errors and losses
Audio content	Delay modes
Object-based bitstream editing and manipulation	Complexity modes
Object-based scalability	Bit-rate modes
Object-based random access and user controls	Downmix [†]
Time scale change	Transcoding
Pitch change	Tandem coding
	Audio formats
	Improved coding efficiency

Table 1.1 MPEG-4 requirements [N4319] (Continued)

Requirements for synthetic audio objects	
Low bit-rate speech	Text to speech
Synthetic speech data	Sound synthesis
Requirements for delivery multimedia integration format (DMIF)	
Connectivity	End-to-end Quality of Service (QoS) management
Transparency	
Application service enablement	Network-based stream processing and management
Requirements for MPEG-J	
Functional requirements	Byte code execution
Byte code delivery	Event mechanism
Authentication	
Requirements for multiuser environments	
Scene graph representation	IPMP and sharing tools
Audiovisual objects and avatars representation	Application programming interfaces
Delivery and stream management	
Requirements for animation framework	
Enhanced texture mapping	Reusability of scene graph nodes and animation streams
Animation support	Persistence
High-level shape representation	Compression of animated objects
Requirements for intellectual property management and protection (IPMP)	
Identification of intellectual property	Intellectual property management and protection interfaces
Intellectual property management and protection hooks	

*Relates to the repeated encoding and decoding (cascading) of audio or video material, either with the same or different codecs.

†Relates to the ability to reduce the number of channels to a configuration with a lower number of channels for presentation purposes (e.g., listening to multichannel audio using stereophonic reproduction).

Although efficient compression was not the only first-priority requirement in MPEG-4 (as it had been for MPEG-1 and MPEG-2), it is clear that it has been a central requirement in MPEG-4 in the sense that, whatever the type of data that had to be represented by binary encoding (e.g., shape for video objects, facial animation parameters for 3D facial models, or even scene composition data), the target was always to reach the smallest number of bits for a certain level of quality. Although some people claim that efficient compression of data is not a must today because of the growing availability of bandwidth, MPEG always acknowledged that bandwidth resources (either for transmission or for storage) are still limited, and thus efficient compression is required. This is even truer for some relevant recent transmission cases, such as the Internet and mobile networks, where bandwidth limitations and efficient compression are major issues. However, in the context of MPEG-4, efficient compression must be balanced against other major requirements, such as those related to interactivity capabilities (which have a price in terms of compression efficiency compared with noninteractive representation schemes), if new functionalities not supported by frame-based coding schemes are to be provided.

As noted, the MPEG-4 requirements' development process has been evolving since the beginning of the standardization process; this evolution has been targeting the inclusion of additional requirements related to functionalities, which fit and complement well the MPEG-4 vision. Because the MPEG-4 standardization process is not yet finished, it cannot be said that all the requirements have been addressed by means of an MPEG-4 tool. However, it is possible to say that all of the not-yet-addressed requirements are either being worked on or should be removed in a short time if the industries do not show sufficient support to move to the technical development phase.

1.1.3 Tools

The major trends mentioned—notably the mounting presence of audiovisual media on all networks, increasing mobility, and growing interactivity—have driven, and continue to drive, the development of the MPEG-4 standard.

To address the identified functionalities and requirements [N4319], a set of tools was developed to perform the following functions [Pere99]:

- ✎ Efficiently represent a number of data types through media codecs
 - ✕ Video from very low bit rates to very high-quality conditions
 - ✕ Music and speech data for a very wide bit-rate range, from transparent music to very low bit-rate speech
 - ✕ Generic dynamic 3D objects as well as specific objects such as human faces and bodies
 - ✕ Synthetic speech and music, including support for 3D audio spaces
 - ✕ Text and graphics

- ☞ Provide fine granularity scalability in the quality, temporal, and spatial dimensions
- ☞ Provide, in the encoding layer, resilience to residual errors for the various data types, especially under difficult channel conditions such as mobile ones
- ☞ Independently represent the various objects in the scene, allowing independent access for their manipulation and reuse
- ☞ Compose audio and visual (natural and synthetic) objects into one audiovisual scene in a synchronized way
- ☞ Describe the objects and the events in the scene
- ☞ Provide interaction and hyperlinking capabilities
- ☞ Manage and protect intellectual property on audiovisual content and algorithms, so that only authorized users have access to the content
- ☞ Provide a delivery-media-independent representation format, to transparently cross the borders of different delivery environments

The major difference with previous audiovisual coding standards, at the basis of the new functionalities, is the object-based audiovisual representation model that underpins MPEG-4 (see Figure 1.1). An object-based scene is built using individual independent objects that have relationships in space and time. This representation approach offers a number of advantages: First, different object types may have different suitable coded representations—for example, a synthetic moving head is best represented using animation parameters, whereas video benefits from a smart representation of pixel values. Second, it allows the harmonious integration of different types of data into one scene, notably with natural and synthetic origins—for example, an animated cartoon character in a real world environment, or a real person in a virtual studio set. Third, interacting with the objects and hyperlinking from them is

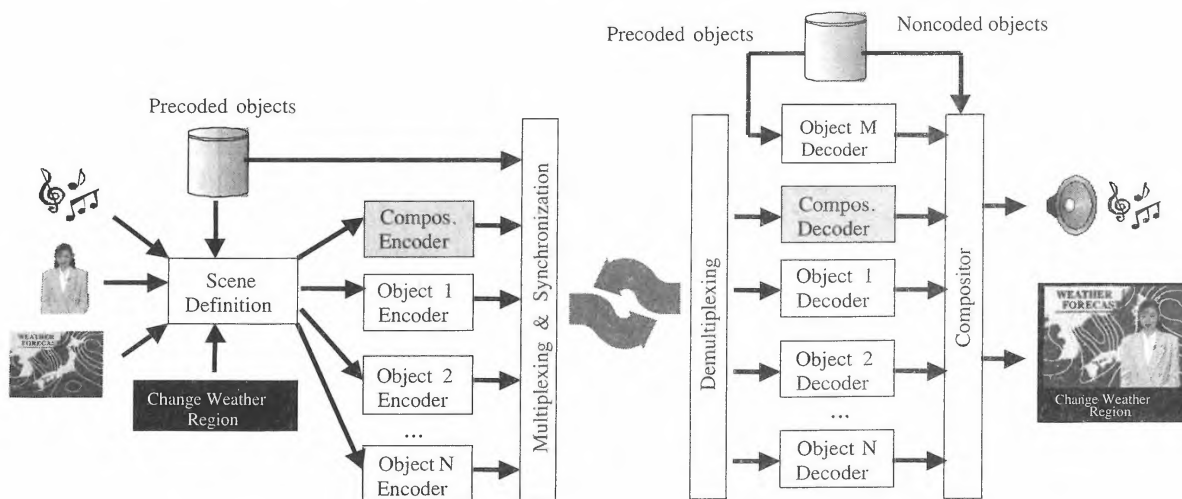


Fig. 1.1 MPEG-4 object-based representation architecture.

now feasible. There are many more advantages—such as the selective spending of bits, the easy reuse of content without transcoding, and the provision of sophisticated coding solutions for scalable content on the Internet—all of them resulting from the adoption of the object-based representation model.

The applications that benefit from the new concepts and functionalities are found in many (and very different) environments [N2724]. Therefore, MPEG-4 is constructed as a toolbox rather than a monolithic standard, using profiles that provide solutions for these different application settings (see Chapter 13). This means that, although MPEG-4 is a standard comprising a vast array of technologies, it is structured so that solutions are available at the measure of the needs. It is the task of each implementer to extract from the MPEG-4 standard the technical solutions adequate to his or her needs (likely a small subset of the standardized tools) by choosing the adequate profiling combination.

Because a standard is always a constraint on freedom, it is important to make it as minimally constraining as possible [Chia97]. To MPEG, this means a standard must offer maximum advantages by specifying the minimum necessary, allowing for competition and evolution of the technology in the *non-normative areas*; these non-normative areas correspond to the tools for which normative specification is not essential for interoperability. On the contrary, normative tools are those defined by the standard whose specification is essential for interoperability. For example, whereas video segmentation and rate control are non-normative tools, the decoding process needs to be normative. The strategy of specifying the minimum for maximum usability ensures that good use can be made of the continuous technical improvements in the relevant technical areas. The consequence is that better non-normative tools can always be used, even after the standard is finalized, and it is possible to rely on competition for obtaining ever-better results. In fact, it is through the non-normative tools that products will distinguish themselves, which only reinforces the importance of this type of tools.

1.1.4 Applications

MPEG-4 wants to address a wide range of applications, many of them completely new, as there are very new functionalities to take benefit from, and many others improved regarding those already available [N2724]. Unlike MPEG-2 where the *killer application* was digital television, in a first approach just understood as the digital translation of the rather old analog version, MPEG-4 does not target a major and exclusive killer application but opens many new frontiers. Playing with audiovisual scenes and creating, reusing, accessing, and consuming audiovisual content will become easier and more powerful. New and richer applications can be developed, for example, in enhanced broadcasting, remote surveillance, personal communications, games, mobile multimedia, and virtual environments. MPEG-4 allows services combining the traditionally different service models: broadcast, (online) interaction, and communication. As such, MPEG-4 addresses *convergence*, under-

stood as the proliferation of audiovisual information in all kinds of services and on all types of (access) networks.

The MPEG-4 Applications document [N2724] describes application examples benefiting from the MPEG-4 technology that will serve as inspiration to the industry to create many more exciting applications. In this sense, MPEG-4 is a technical playground where many application constructions may be built by the manufacturers and service providers. The MPEG-4 Applications document suggests the following applications, using both audio and visual information or just one of them: broadcast, collaborative scene visualization, content-based storage and retrieval, digital amplitude modulation (AM) broadcasting, digital television set-top box, DVD, infotainment, mobile multimedia, real-time communications, streaming video on the Internet/Intranet, studio and television postproduction, surveillance, and virtual meetings.

1.2 FORMAL STANDARDIZATION PROCESS

MPEG (Moving Picture Experts Group) formally called *Coding of Moving Picture and Audio* is Working Group 11 (WG11) of Subcommittee 29 (Coding of Audio, Picture, Multimedia and Hypermedia Information) of the ISO/IEC (International Organization for Standardization/International Electrotechnical Commission) Joint Technical Committee 1 (JTC1) [MPEG]. JTC1's scope is standardization in the field of information technology: According to JTC1 [JTC1], information technology includes the specification, design, and development of systems and tools dealing with the capture, representation, processing, security, transfer, interchange, presentation, management, organization, storage, and retrieval of information.

Since the major objective of MPEG is to produce technical specifications, it is important to have an idea, even if brief, about the types of documents that can be produced by a group like MPEG. Through JTC1, MPEG can produce the types of documents presented in the following:⁵

- ✎ **International standards:** These are documents with the technical specification of the standard. After the first edition of a standard, new editions may be published, notably to incorporate in one document the previous edition and all the amendments and corrigenda meanwhile issued. Whenever a new edition for a certain Part of the standard includes new technology (and thus is not just a compilation of previously approved technology), it must be approved following a process similar to amendments.

5. For each type of document, the various successive stages, as well as the minimum time the voting period by national standardization bodies may take for each stage (this means the minimum time to take the promotion of a certain document from one stage to the next), are indicated.

MPEG-4 Overview

by Olivier Avaro, Rob Koenen, and Fernando Pereira

Keywords: MPEG-4 design goals, multimedia architecture, audiovisual object representation, MPEG-4 tools, multimedia standards, MPEG-4 applications

The MPEG-4 project started in 1992 with the goal of setting a standard for *Very Low Bit Rate Audiovisual Coding* [N271]. The initial participants were companies from the consumer electronics, computer, and telecom industries, as well as academia. There was the promise of great new coding methodologies that would greatly improve (video) compression efficiency over the existing hybrid DCT-based coding schemes. The participants discussed whether MPEG-4 should be centered on *Very Low Bit Rates* or on *Very High Compression Efficiency*. Unfortunately, the “Great Plan” was built on quicksand, which became apparent in July of 1994 at the Grimstadt MPEG meeting. At that point, MPEG altered the goals of MPEG-4 considerably. After reassessing developments in the multimedia arena, MPEG redefined the goals to align them with the anticipated evolution in the production, delivery, and consumption of audiovisual content, as summarized in Table 2.1. The *linear* approach that had dominated the multimedia industry at all stages of the content lifecycle was replaced by a *nonlinear, object-based, interactive* way of dealing with audiovisual data. As a consequence, to reflect the focus of the work more accurately, the name of the MPEG-4 project was changed to *Coding of Audio-Visual Objects* [N998]. The scope of the MPEG-4 standard is defined in more detail in the MPEG Requirements document [N4319]. At the time of writing (February 2002), the MPEG-4 standard was

Table 2.1 Evolution of multimedia content production, delivery, and consumption

	Traditional	MPEG-4
Production	• Mostly 2D content • Natural content, produced with cameras and microphones • Composition (mixing) of objects at production stage	• 2D and 3D content • Content with hybrid natural and computer-generated audio and video • Rectangular and presegmented video content • Explicit coding of composition of separate objects
Delivery	• Few networks carry audiovisual information (satellite, LAN, ISDN) • Many networks have their own content representation scheme • Communications and delivery over homogeneous networks	• Virtually all networks carry audiovisual information • Common content representation scheme • Communications and delivery over heterogeneous networks (multiple network types)
Consumption	• Mostly passive content • Content precomposed	• Composition of objects at consumption stage • Information is read, seen, and heard in an interactive and personalized way • More and more information is audiovisual and digital

largely finalized, although some additional tools for advanced functionality were still under development. A complete overview of the MPEG-4 technology is provided in the MPEG-4 Overview [N4505], which is updated at every MPEG meeting—about once every quarter.

2.1 DESIGN GOALS

The main MPEG-4 design goals, as formulated in 1994, can be summarized as follows:

1. To provide the technological foundations to represent multimedia content for a wide range of multimedia services and networks. To provide a common representation basis encompassing the interactive, broadcast, and conversational paradigms. To allow for seamless content usage across these different types of services.
2. To extend the interactivity found in text and images to content having a temporal dimension. To provide for this type of content client-side interactivity (i.e., without the need for a return channel), as well as server-side interactivity, exploiting an on-line connection.

3. To extend multimedia content usage from high-bandwidth networks to all types of networks, including those with low bandwidth and high error rates. MPEG calls this *Universal Access*.
4. To integrate different media types into a single framework, giving each its own optimized coded representation.
5. To offer rich interactivity to content authors and content users.
6. To allow for consumption of the audiovisual content that respects the usage rights that are attached to it.

For all its complexity and power, the MPEG-4 standard is based on a very simple concept: an audiovisual scene is composed of audiovisual objects whose properties are defined using a scene description language. This concept allows

- ☞ Interaction with individual elements in the audiovisual content, named *audiovisual objects*
- ☞ Adaptation of the coding scheme on a per-object basis
- ☞ Easy reuse and customization of the audiovisual content

Audiovisual objects can differ in nature. They can be audio-only objects like a multichannel audio content or speech, or video-only objects, like traditional rectangular movies or arbitrarily shaped video objects. Objects can be natural, like audiovisual data captured from a microphone or from a camera, or synthetic, like text and graphics overlays, animated faces, or synthetic music. They can be 2D, like a Web page, or 3D, such as spatialized sound and 3D virtual worlds.

The scene description provides the spatial and temporal relationships between the audiovisual objects. These relationships can be purely 2D or 3D, or a mixture of 2D and 3D (for instance, a 2D “subscene” mapped in a 3D virtual world). The scene description includes commands specifying the behavior and interactive options embodied in the audiovisual objects and scenes. In addition, MPEG-4 specifies protocols to modify and animate the scene over time (for instance, adding or deleting objects, or changing objects’ properties and behavior). The scene description is dynamic, which allows incremental build-up of the scene (as opposed to waiting for a download and then playing).

All information representing the audiovisual objects and the audiovisual scene description are provided as compressed binary streams, and all of these streams can be synchronized.

A possible audiovisual scene including natural and synthetic objects is shown in Figure 2.1. This figure shows how various visual and audio streams are composed on the top of a fixed still picture background (an object in its own right), according to a scene description stream. The diagram introduces the concept of *object descriptor* (OD). An object descriptor is a structure that lays the links between the scene description and the individual streams with media content. More details on these object descriptors may be found in Chapter 3.

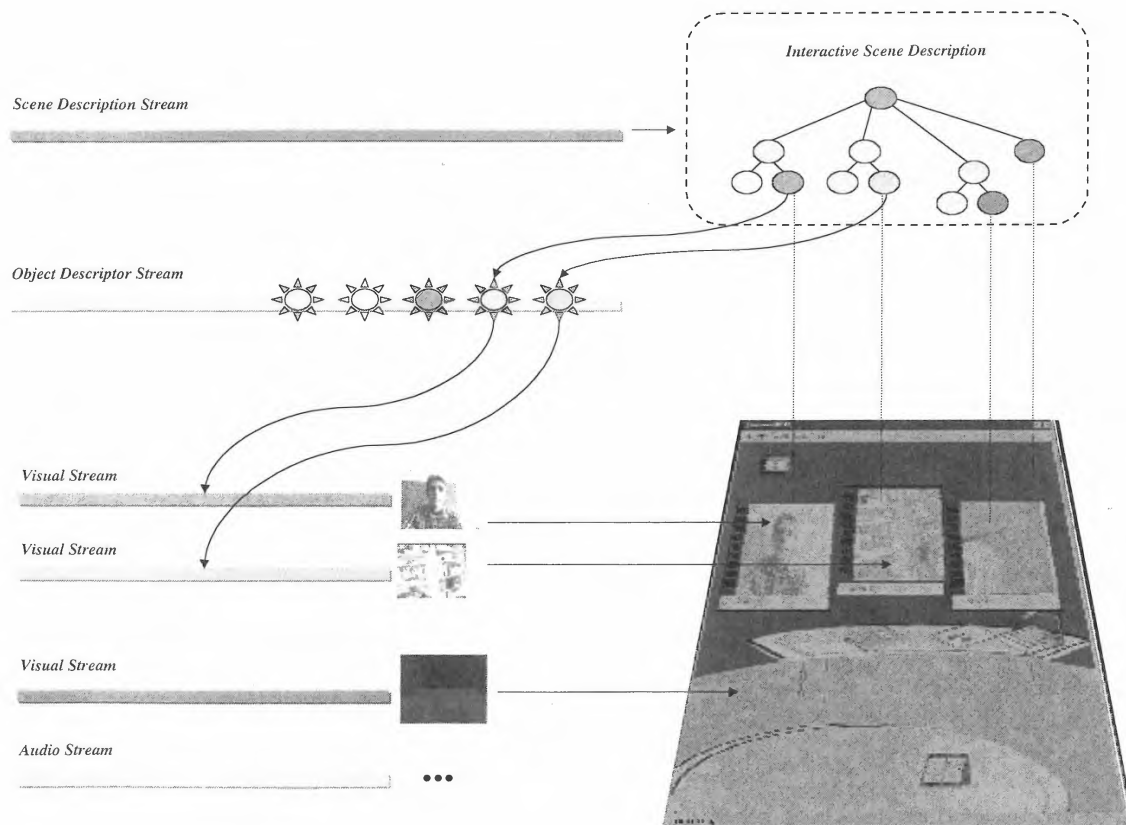


Fig. 2.1 Interactive audiovisual scene with objects.

2.2 AN END-TO-END WALKTHROUGH

This section explains the high-level workings of the MPEG-4 standard using an end-to-end walkthrough of an MPEG-4 session. Although the MPEG-4 standard is designed to be used in many environments, the way MPEG-4 content is produced, delivered, and consumed often follows the same path. Such a typical path is shown in Figure 2.2, which highlights the main interoperability elements of the MPEG-4 standard as well as their respective position in the chain. While this walkthrough provides a typical scenario, MPEG-4 also can be used in other scenarios that are not covered by the walkthrough in Figure 2.2.

At the start of the walkthrough are content authors, who produce audiovisual content with the authoring tools of their preference. Part of the content creation process may be live; needless to say, though, off-line creation is also important in many MPEG-4-based applications. The content creation process can be separated into two major steps: authoring and publishing. Authoring involves the production of the audiovisual data, including the media, the scene description, and the interaction. Publishing involves adapting the content to the constraints imposed—by networks and terminals, for example.

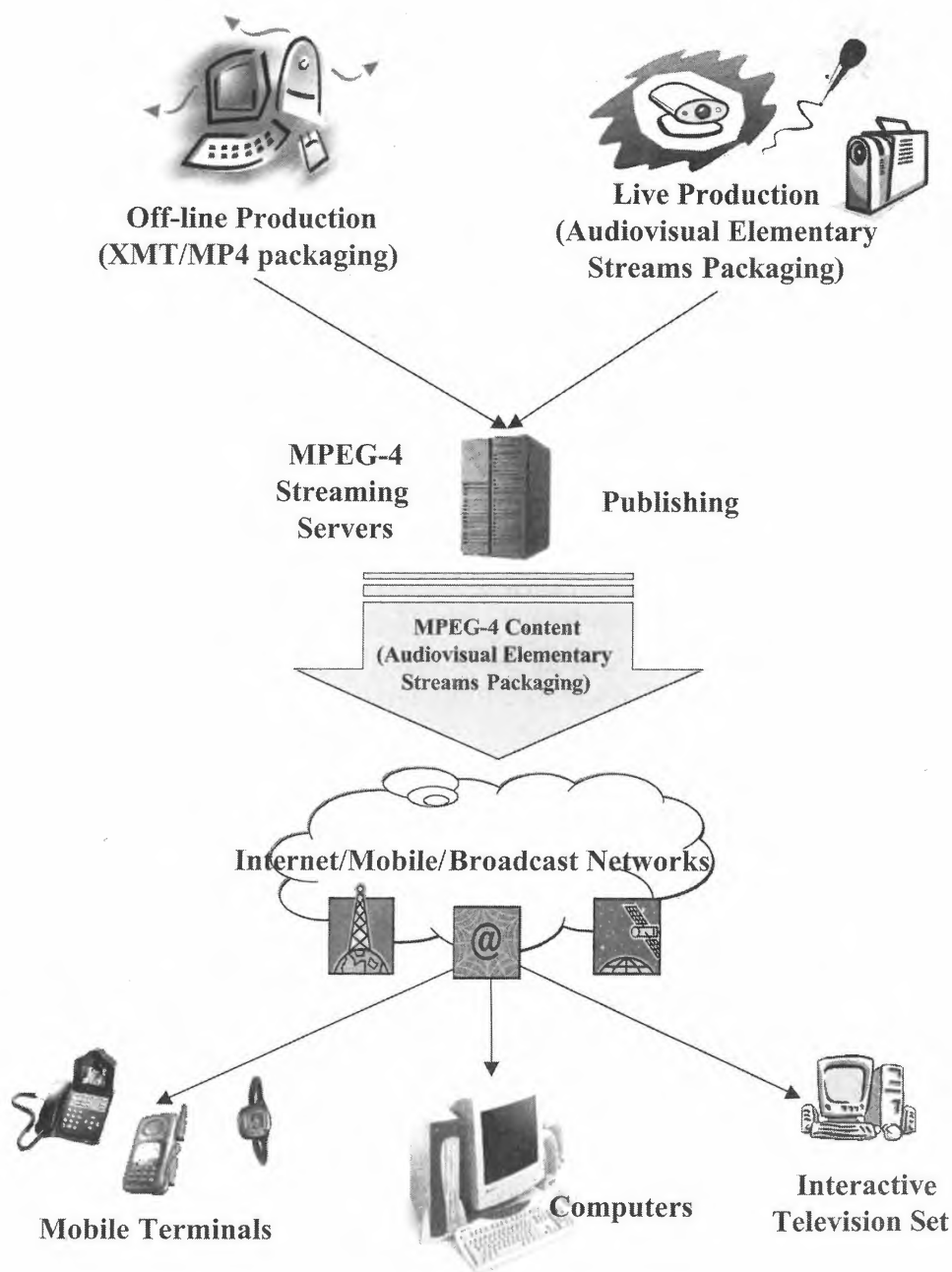


Fig. 2.2 Example MPEG-4 end-to-end walkthrough.

While streaming content is produced using MPEG-4's binary streaming formats, off-line multimedia content can be delivered to servers in an interoperable way using the MPEG-4 eXtensible MPEG-4 Textual format (XMT), as described in Chapter 6, or using the binary MP4 file format, as described in Chapter 7. The choice between these two depends on the freedom that authors want to offer users (or other authors) at the next stage of the delivery chain.

XMT provides a structured and extensible representation of the scene description, allowing much flexibility for further manipulation or adaptation of this data (e.g., for automated processing like customization). It may contain additional information such as metadata, for instance, to express constraints on the adaptation to the delivery network. This makes it an appropriate format for exchanging content that needs to be further processed; XMT is therefore useful for the exchange of content between authors.

MP4 is more rigid than XMT in the sense that the scene description data is binary coded and hence hard to alter afterwards. The binary form is obviously more compact but also more deterministic with regard to what the users will see and hear, which is useful to ensure that an author's creations are presented as intended.

While MP4 and XMT files are the natural interoperability formats between authoring tools and MPEG-4 servers, this does not mean that the content will be stored as MP4 or XMT files on the server. Server implementations may have other ways to represent the content, optimized for specific software and hardware. What matters is that the served streams be compliant with the standard. But MPEG-4 servers can also directly use MP4 or XMT files to serve content on various networks. If XMT is used, the server needs to perform more than streaming; it will also need to encode the scene description data into binary form.

What goes out of the server are streams of data containing MPEG-4 content named *elementary streams*. The content of these elementary streams is discussed later in this chapter. What is important at this stage is that MPEG-4 audiovisual scenes can be split in several elementary streams, that these streams can be carried on different networks and that end terminals receiving these streams from different networks are capable of reconstructing the transmitted data in a synchronized manner. When content is produced live, it can be delivered to servers as interoperable elementary streams.

One of MPEG-4's design goals is to cover a wide range of access conditions, so that content can be created once and played on any network. This is achieved by abstracting the content delivery layer with an interface as specified in the Delivery Multimedia Integration Framework (DMIF) corresponding to MPEG-4 Part 6 [MPEG4-6]. This interface is named the DMIF Application Interface (DAI).

When streaming content, the MPEG-4 interoperability points are the specific formats of (and constraints on) the individual elementary streams, and the compliance to the walkthrough (i.e., the series of steps that has to be performed to access the elementary streams) defined by the DAI. In principle, what happens below the DAI is outside the scope of the MPEG-4 standard. In this way, the multimedia representation can be kept completely independent from the way it is delivered. There are exceptions to this rule: an efficient low-complexity multiplexing tool (FlexMux) and a dedicated file format (MP4) were developed because MPEG-4 needed specific tools for its transport (see Chapter 7).

In order to transport the MPEG-4 content in existing environments, network-specific transport mechanisms have been defined. Transport in MPEG encompasses both wire formats and file formats. Currently, these transport mechanisms are available:

- ☞ Storage of MPEG-4 content in MP4 files [MPEG4-1]
- ☞ Carriage of MPEG-4 content on the Internet [MPEG4-8, GENT01]
- ☞ Carriage of MPEG-4 content in MPEG-2 transport streams [MPEG2-1]

These mechanisms are described in detail in Chapter 7. The spectrum of MPEG-4 consumption devices includes desktop computers, mobile devices, and interactive television sets. This last device nicely illustrates the various ways MPEG-4 content can be consumed.

Assume that the interactive TV set in Figure 2.2 receives through a satellite connection an MPEG-2 transport stream containing several MPEG-2 digital TV programs. Using the MPEG-2 extension that specifies the carriage of MPEG-4 on MPEG-2 transport streams [MPEG2-1], MPEG-4 content related to such a TV program can be sent along with the linear programming. This can provide the user with an enhanced interactive experience, such as a local interaction with a 3D model of a car in an ad or a multimedia electronic program guide. Note that all the interactivity is sent in the broadcast stream and a return path is not required.

Assume now that the TV set is also connected to the Internet. It is then possible to augment the broadcast experience with client-server functionality as well as to send rich media that can be blended with the basic TV program. One can imagine a range of services encompassing program enhancements with video clips streamed from the Internet on-demand up to multiuser games related to the TV programs with votes, 3D chats, and interaction with the scenario of the broadcast content.

2.3 TERMINAL ARCHITECTURE

The overall architecture of an MPEG-4 terminal is depicted in Figure 2.3. At the bottom of the figure is the storage or transmission medium; this refers to the lower layers of the delivery infrastructure. MPEG-4 data can be transported on a variety of delivery systems as described in Section 2.2. This includes MPEG-2 transport streams, RTP/UDP over IP, and MPEG-4 (MP4) files. MPEG-4 data can also be sent over other transport mechanisms, such as Asynchronous Transfer Mode (ATM) or Digital Audio Broadcast (DAB), using one or a combination of the previous stacks of protocols.

Most of the currently available transport layer systems provide native means for multiplexing information. There are, however, a few exceptions, such as RTP Internet connections. In addition, the existing multiplexing mechanisms do not always fit MPEG-4 needs in terms of low delay, or they

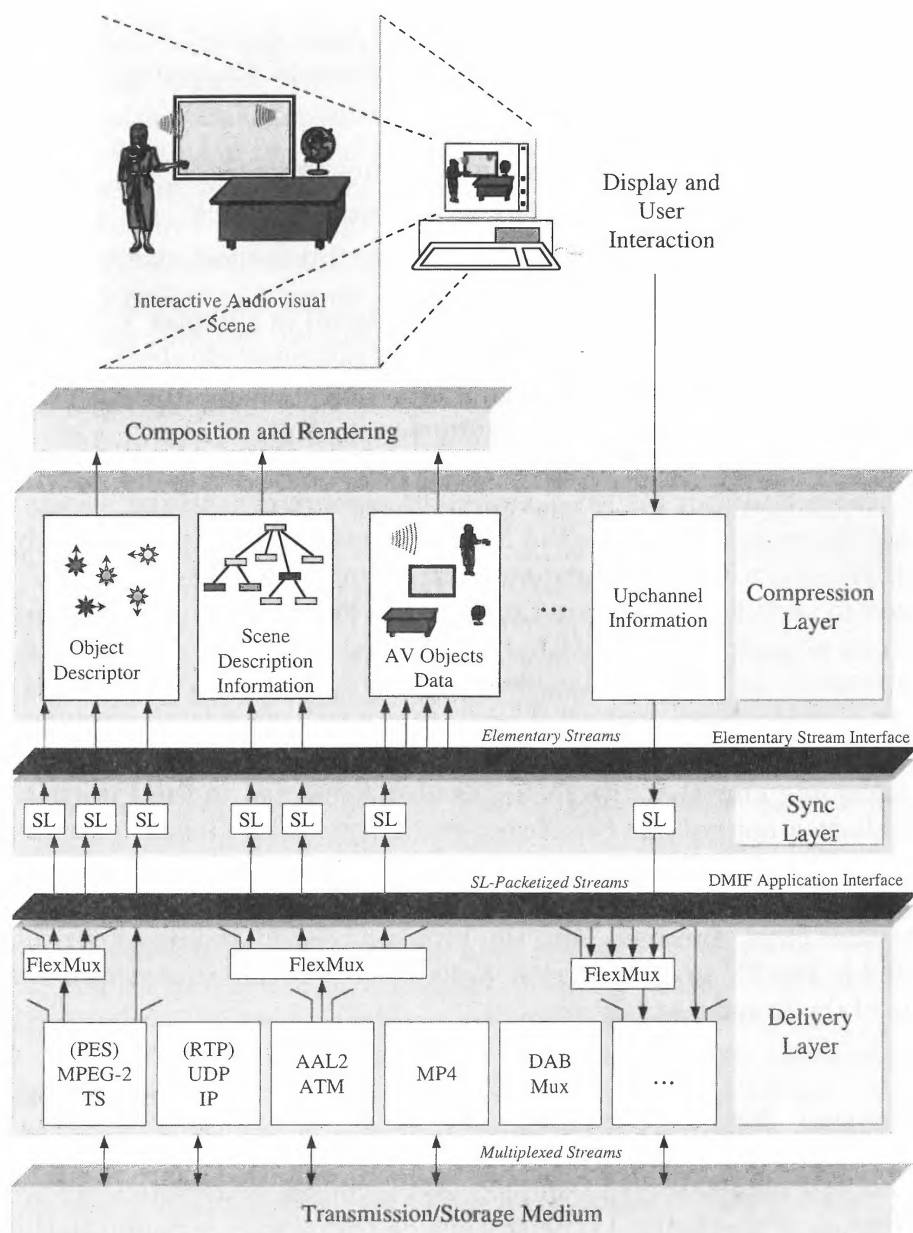


Fig. 2.3 MPEG-4 terminal architecture [MPEG4-1].

may incur prohibitive overhead in handling the expected large number of streams associated with an MPEG-4 session. As a result, MPEG-4 has defined a multiplexing tool, FlexMux, that can be used on top of the existing transport delivery layer (see Chapter 7).

The delivery layer provides a number of elementary streams to the MPEG-4 terminal. These streams can be audiovisual objects data, scene

description data, control data in the form of object descriptors, or metadata describing the content or associating intellectual property rights to it.

Note that not all the streams have to be downstream (server to client); upstream channels can also exist, for the purpose of conveying data from the terminal to the transmitter or server. MPEG-4 standardizes the mechanisms that trigger the transmission of such upstream data at the terminal, as well as the formats in which they are transmitted back to the sender.

It is important that there be a common mechanism for conveying timing and framing information that applies to all types of elementary streams. The basic MPEG-4 timing model and the Sync Layer (SL) provide this functionality (see Chapter 3). The SL is a flexible and configurable packetization tool that allows the inclusion of timing, fragmentation, and continuity information in the associated data packets. Such information is attached to data units called *access units*, which comprise complete presentation units (e.g., an entire video frame or an audio frame).

Elementary streams are sent to their respective decoders, which process the data and produce composition units (e.g., a decoded video frame or, in MPEG-4 terminology, a Video Object Plane [VOP]). In order for the receiver to know what type of information is contained in each stream, control information in the form of ODs is used. The ODs describe the properties of elementary streams and are used to associate one or more elementary streams to each of the audio and visual objects in the scene. Through these ODs, the terminal can identify the content being delivered to it. Without an OD, the terminal wouldn't know how to use the elementary stream. ODs may also contain—or point to—Object Content Information (OCI). OCI elements can contain data such as content classification, rating, language, and keywords. (Because it was ready to use first, OCI forms sort of a precursor to MPEG-7.) MPEG-4 scenes can also contain MPEG-7 streams through a dedicated elementary stream type. Detailed descriptions of synchronization in MPEG-4 and of the OD framework are given in Chapter 3.

Advanced synchronization mechanisms (described in Chapter 4) augment the basic MPEG-4 timing model to permit synchronization of multiple streams and objects that can originate from a number of sources. The Flextime toolkit allows the definition of simple temporal relationships among MPEG-4 objects, such as CoStart, CoEnd, and Meet, as well as the specification of constraints for the timing relationship between MPEG-4 objects, as if the objects were on springs. The duration of an object is represented by the length of the spring, and these springs can be expanded or contracted as needed, to adjust the duration of each of the objects (adjusting the playback speed). Flextime is modeled after the W3C's Synchronized Multimedia Integration Language (SMIL) [SMIL01], and one of the reasons to include it in MPEG-4 was to provide compatibility with SMIL content.

In the context of an MPEG-4 audiovisual scene, at least one of the streams must contain the scene description information associated with the content. The scene description information defines the spatial and temporal

position of the various objects, their dynamic behavior, and any interactivity features embedded in the content. As mentioned above, the (coded) data that represents an audiovisual object is carried in separate elementary streams. The scene description contains *pointers* to ODs when it refers to a particular audiovisual object. The MPEG-4 scene description language, called *binary format for scenes* (BIFS) [MPEG4-1], is further explained in Chapter 4. It complements the OD framework that defines the objects and their characteristics.

A key feature of the scene description is that, since it is carried in its own elementary stream(s), it can contain full timing information. This implies that the scene can be dynamically updated over time, a feature that provides considerable power for content creators. There is also a special lightweight mechanism to animate parts of the scene: *BIFS Anim*. This animation is accomplished by coding, in a separate stream, only the parameters that need to be updated.

The system's compositor uses the scene description information to aggregate the various natural and synthetic audiovisual objects and render the final scene for presentation to the user. MPEG-4 natural video objects (see Chapter 8) include rectangular and shaped, scalable and error-resilient video as well as still pictures [MPEG4-2]. Synthetic visual objects can be 2D and 3D meshes, as well as animated 3D faces and bodies as described in Chapter 9 and specified in both MPEG-4 Visual [MPEG4-2] and MPEG-4 Systems [MPEG4-1].

The palette of the MPEG-4 Audio tools includes speech coding from 2 to 24 kbit/s (see Chapter 10), general audio coding from 4 to 64 kbit/s/channel and above (see Chapter 11), as well as synthetic audio such as text-to-speech synthesis and synthetic music coding (see Chapter 12). All these tools are specified in MPEG-4 Audio [MPEG4-3], and the interfaces exposed at the scene description level are specified in MPEG-4 Systems [MPEG4-1].

The scene description tools provide mechanisms to capture user or system events. In particular, these mechanisms allow the association of events to user operations on desired objects and they can—in turn—modify the behavior of the object. Event processing is the core mechanism with which application functionality and differentiation/customization can be provided. In order to provide flexibility in this respect, MPEG-4 allows the use of ECMAScript (also known as JavaScript) scripting within the scene description. Use of scripting tools is essential in order to access state information and implement sophisticated interactive applications.

MPEG-4 also defines a set of Java language Application Programming Interfaces (APIs), generally called MPEG-J, through which access to an underlying MPEG-4 player can be provided to Java applets (called *MPEGlets*). These MPEGlets can form the basis for very sophisticated applications, opening up rich possibilities for audiovisual content creators to author attractive content. Chapter 5 provides a complete description of the MPEG-J application engine.

In addition to the new interactive features for end users, the object-based content structure provides great benefits to content creators as well. The use of an object-based structure, where composition happens at the receiver, con-

siderably simplifies the content creation process. Starting from a set of audio-visual objects, it is easy to define a scene description that combines these objects in a meaningful presentation. An essentially similar approach is used in HTML and Web browsers, which makes it easy for nonexpert users to create their own content. The fact that the content's structure survives the process of coding and distribution also allows for the reuse of the objects in the scene. For example, content filtering and searching applications can be easily implemented using ancillary OCI carried in object descriptors. Finally, when allowed by the content author, consumers with the right tools can easily extract and reuse individual objects.

2.4 MPEG-4 TOOLS

This section provides a brief overview of the tools available in the MPEG-4 standard. A more detailed description can be found in the MPEG-4 Overview [N4505]. Each of these tools is explained in detail in one of the remaining chapters of this book.

The tools are organized in one of four categories—Systems, Visual, Audio, and DMIF—corresponding to the four technology specification Parts of the MPEG-4 standard. The tools listed cover the state of the standard as of February 2002. Since that date, more tools may have been standardized in MPEG-4; for up-to-date information, consult the MPEG Web site, *mpeg.telecomitalia.com* [MPEG].

2.4.1 Systems Tools

The major tools standardized in MPEG-4 Systems [MPEG4-1] are

- ☛ **Object descriptor framework:** The OD framework defines the relationship between the individual elementary streams and the media objects in the scene (e.g., the audio or video streams associated with an object representing a participant in a videoconference). ODs provide information such as the elementary streams available to represent a given media object, the characteristics of the decoders needed to consume the elementary streams, and the location (possibly a URL) of the elementary streams data. ODs include some *Intellectual Property Management and Protection* (IPMP) information in the form of data sets covering the identification of intellectual property rights associated with the media objects. See Chapter 3 for further detail.
- ☛ **Systems decoder model:** The systems decoder model provides the basic synchronization and streaming features of the MPEG-4 standard. The model defines the initialization and continuous management of the receiving terminal's buffers. It also defines the timing-identification, synchronization, and recovery mechanisms. For further details see Chapter 3.

- ☞ **Binary format for scenes:** BIFS describes the spatio-temporal arrangements of the MPEG-4 objects in the scene. The scene description provides a rich set of functionality for 2D and 3D composition as well as text and graphics primitives. It also provides forms of interactivity (e.g., general event handling and routing between objects in the scene) among events triggered by the user or the scene, as well as client- and server-based interaction. See Chapter 4 for details.
- ☞ **MPEG-J:** MPEG-J allows the use of Java programs to complement the logic and programmatic part of MPEG-4 content. These programs are called MPEGlets when they are transmitted along with the multimedia content. MPEG-J defines interfaces to various aspects of the terminal and networks in the form of Java APIs. MPEG-J also defines a delivery mechanism allowing MPEGlets and other Java classes and objects to be streamed separately. Chapter 5 provides greater detail.
- ☞ **Extensible MPEG-4 textual format:** XMT is a framework for representing MPEG-4 scene descriptions using a textual syntax. The XMT format allows the content authors to exchange their content with other authors or with tools or service providers, and it facilitates interoperability with both the X3D format being developed by the Web3D consortium [X3D01] and the Synchronized Multimedia Integration Language (SMIL) format from the W3C consortium [SMIL01]. See Chapter 6 for more detail.
- ☞ **Transport tools:** Although transport of multimedia content is not in the core activity of the standard, MPEG-4 defines two transport tools, MP4 and FlexMux, for which there was no clearly identified standardization body where they could have been standardized. MP4 is a tool for storing MPEG-4 data in a file (the MPEG-4 file format). FlexMux is a tool for the interleaving of multiple streams into a single stream, including timing information. See Chapter 7 for a discussion of these tools.
- ☞ **IPMP “hooks”:** APIs that provide an interface to IPMP systems and a way to identify which IPMP system was used. The IPMP system itself is not specified by MPEG. At the time of writing (February 2002), MPEG-4 was developing an extension to these MPEG-4 hooks that will provide more digital rights management capabilities as well as more flexibility for the IPMP system [N4270]. The goal of this work is to allow end users to consume protected content in a transparent manner and to provide renewability of IPMP systems (see Chapter 3).

2.4.2 Visual Tools

The major tools standardized in MPEG-4 Visual [MPEG4-2] are introduced here.

- ☞ **Video compression tools:** Algorithms providing the efficient compression of video for bit rates between 5 kbit/s and 1 Gbit/s and for resolutions from sub-QCIF to studio editing resolutions (4k×4k pixels).

Progressive and interlaced video are supported. Compact coding of textures with a quality adjustable from *acceptable* for very high compression ratios up to *near lossless* is supported, notably targeting texture mapping on 2D and 3D meshes. Random access of video provides functionalities such as pause, fast forward, and fast reverse of stored video. Different types of scalability are provided, including complexity scalability at the encoder and decoder, object scalability, spatial scalability, temporal scalability, and quality scalability. See Chapter 8 for more detail.

- ✎ **Robustness in error-prone environments:** Error resilience is supported to assist the access of images and video over a wide range of storage and transmission media. This includes the useful operation of image and video compression algorithms in error-prone environments at low bit rates (e.g., less than 64 kbit/s). There are coding tools that address both the bandwidth-limited nature and error-resilience aspects of wireless networks. See Chapter 8 for more detail.
- ✎ **Fine grain scalability (FGS):** FGS allows small quality steps by adding or deleting layers of extra information with fine granularity (quality scalability). It is useful in a number of environments, notably for streaming purposes, but also for dynamic (*statistical*) multiplexing of pre-encoded content in broadcast environments. See Chapter 8 for more detail.
- ✎ **Shape and alpha channel coding:** Efficient techniques are provided for coding of *binary* and *gray-scale* or *alpha* shape information. A binary alpha map defines whether a pixel belongs to an object; in a binary alpha map, a pixel can be *on* or *off*. A gray-scale map offers the possibility of defining a certain transparency for each pixel. See Chapter 8 for more detail.
- ✎ **Face and body animation:** The face and body animation tools specify parameters for defining, calibrating, and animating synthetic faces and bodies. The 3D models themselves are not standardized by MPEG-4; only the calibration and animation parameters are. See Chapter 9 for more detail.
- ✎ **Coding of 2D meshes:** MPEG-4 has tools for mesh-based coding and animated texture transfiguration. These tools comprise 2D Delaunay and regular mesh formalisms, with motion tracking of animated objects, motion prediction, and suspended texture transmission with dynamic meshes. There is also geometry compression for motion vectors (2D mesh compression with implicit structure and decoder reconstruction). See Chapter 9 for more detail.
- ✎ **Coding of 3D meshes:** MPEG-4 includes tools for coding 3D polygonal meshes. Polygonal meshes are widely used as a general representation of 3D objects. The underlying technologies compress the connectivity, geometry, and properties such as shading normals, colors, and texture coordinates of 3D polygonal meshes. See Chapter 9 for more detail.

Natural video coding tools are presented in further detail in Chapter 8, and synthetic coding tools are described in Chapter 9.

2.4.3 Audio Tools

This section lists MPEG-4's tool set for audio coding. Speech and general audio coding tools are explained in more detail in Chapters 10 and 11, respectively; tools for coding of synthetic audio and speech are described in Chapter 12. The major tools standardized in MPEG-4 Audio [MPEG4-3] are

- **Speech coding:** MPEG-4 Audio supports speech coding at bit rates from 2 kbit/s up to 24 kbit/s, using two algorithms: a Code Excited Linear Prediction (CELP) coder and a parametric coder. With variable bit-rate coding, even lower bit rates can be achieved—down to an average of 1.2 kbit/s. Low delay is possible for communications applications. The parametric Harmonic Vector Excitation Coding (HVXC) coder allows the user to modify speed and pitch during playback. If the CELP coder is used, a change of the playback speed can be achieved by using an additional tool for effects processing. See Chapter 10 for more detail.
- **General audio coding:** Most of the MPEG-4 general audio coding tools are based on the filterbank coding approach as used in MPEG-2 Advanced Audio Coding (AAC) [MPEG2-7]. Building on this technology, MPEG-4 added new tools with the target of improving the compression performance and supporting new functionalities. For example, the Transform-domain Weighted Interleave Vector Quantization (TwinVQ) coder supports operation at very low bit rates (e.g., 6 kbit/s/channel). MPEG-4 is the first audio coding standard to support scalability modes (e.g., fine grain scalability with a granularity down to 1 kbit/s/ch). A special low delay coding mode is available for communication applications among others. MPEG-4 also specifies a parametric audio coder, Harmonic and Individual Lines plus Noise (HILN), that makes a decomposition of the input signal into distinct sound components, namely a harmonic tone, individual sinusoids, and a noise component. This coder can operate at bit rates down to about 4 kbit/s, provides bit-rate scalability, and can be combined with the MPEG-4 HVXC parametric speech coder. See Chapter 11 for more detail.
- **Synthesized audio coding:** Structured audio coding tools provide a powerful framework in which score-based control information can be used to control “musical instruments” described with a special language. If limited resources are available, synthetic sound can be generated using a standardized wavetable format. The popular MIDI format is also supported. See Chapter 12 for more detail.
- **Synthesized speech coding:** Text-to-Speech (TTS) tools allow a text, possibly with prosodic parameters (pitch contour, phoneme duration, and so on), to be used to generate intelligible synthetic speech. Scalable TTS cod-

ers' bit rates range from 200 bit/s to 1.2 kbit/s. This tool also enables the prosody of the original speech to be used; lip synchronization control with phoneme information; trick mode functionality; international language and dialect support for the text; international symbol support for phonemes; support for specifying age, gender, and speech rate of the speaker; as well as support for conveying Facial Animation Parameter (FAP) bookmarks.

2.4.4 DMIF Tools

The major tools standardized in DMIF, Part 6 of the MPEG-4 standard [MPEG4-6] are

- ✦ **DMIF-Application Interface (DAI):** The DAI is a transparent application interface allowing access to multimedia content, irrespective of whether the peer is a remote interactive peer, a broadcast, or a local storage entity. The communication model underlying these APIs allows for continuous Quality of Service (QoS) monitoring, specific QoS queries, and notification of QoS violations. It also enables the simultaneous access, presentation, and synchronization of MPEG-4 content carried through different delivery technologies.
- ✦ **DMIF Signaling Protocol:** The DMIF Signaling Protocol is a generic session-level protocol designed for streaming multimedia data. The protocol is used to configure the delivery protocol stacks at the peer ends. The protocol stack may also include the FlexMux tool specified in MPEG-4 Systems [MPEG4-1]. The protocol was conceived with an eye to the future of networking technologies, and it enables features that are not readily available with current techniques. Such features include QoS and resource management and operation over a variety of networks, including heterogeneous networks.

See Chapter 7 for further details on DMIF tools.

2.4.5 Other MPEG-4 Tools

Strictly speaking, the following tools are not part of the MPEG-4 standard, but they allow the transport of MPEG-4 content on various delivery networks. See Chapter 7 for details.

- ✦ **Carriage of MPEG-4 on MPEG-2 Systems:** This extension to the Systems Part of the MPEG-2 standard [MPEG2-1] specifies how MPEG-2 Systems streams may carry individual MPEG-4 audio and visual elementary streams as well as audiovisual scenes with their associated streams. A distinction is made between MPEG-4 individual elementary streams (only MPEG-2 Systems tools are used) and an MPEG-4 audiovisual scene with its associated streams (MPEG-4 Systems stream management tools are used along with MPEG-2 Systems tools).

- ☞ **Carriage of MPEG-4 on IP networks:** This specification of the Internet Engineering Task Force (IETF) describes a payload format for transporting MPEG-4 encoded data using the Real-Time Transfer Protocol (RTTP) [GENT01]. This specification is complemented by Part 8 of the MPEG-4 standard, “Carriage of ISO/IEC 14496 contents over IP networks” [MPEG4-8], which provides a framework for the carriage of MPEG-4 content over IP networks as well as guidelines for designing payload format specifications for the detailed mapping of MPEG-4 content into several IP-based protocols.

2.4.6 Profiles and Levels

MPEG-4 would be too unwieldy for most applications if any compliant decoder had to be able to support all of the tools listed above. Therefore, MPEG has defined profiles and levels for the MPEG-4 standard, much like it did for its predecessors MPEG-1 Audio (which has three *layers* of audio coding) and MPEG-2 Audio and Video.

Profiles define subsets of tools that can be used for a large class of applications and services. Profiles are defined for specific tool dimensions (e.g., visual or audio) and not for combinations of these dimensions. This means that application developers can choose the tools they need by picking, for each tool dimension, the profile that best suits the application. Profiles exist for these dimensions:

- ☞ Visual
- ☞ Audio
- ☞ Graphics
- ☞ Scene graph
- ☞ Object descriptors
- ☞ MPEG-J

Levels further limit the complexity of the tools within a given profile (see Appendixes A–D). Conformance points of the MPEG-4 standards are defined as a given profile at a given level. Conformance to the standard can be tested at these conformance points for the various profiling dimensions. Conformant terminals for a given profile and level will interoperate. MPEG-4 profiles and levels are further described in Chapter 13.

2.5 MPEG-4 AND OTHER MULTIMEDIA STANDARDS

The multimedia standards and solutions in place before the MPEG-4 standard was developed influenced the development of MPEG-4. Among the standards and solutions that MPEG-4 has used and referenced, the following are especially relevant:

Object Description and Synchronization

by Carsten Herpel

Keywords: object description, stream description, OCI, IPMP, content access, synchronization, system decoder model

An MPEG-4 presentation may consist of many audiovisual objects that are conveyed through a number of audiovisual streams to a receiver, either in a truly streaming manner or stored in a file. An important task in structuring an MPEG-4 presentation is the description of all streams, including information such as media codecs, bit rates, and so on used to encode them as well as the relations between streams. Additional information about streams may be desired for semantic description or for implementing digital rights management. Further to the stream description, means for a synchronized presentation of all streams are required, including a timing model and a buffering model for the MPEG-4 terminal. The way MPEG-4 provides all these functionalities is presented in this chapter. These tools are specified in MPEG-4 Part 1: Systems [MPEG4-1].

3.1 OBJECT DESCRIPTORS: ENTRY POINTS TO MPEG-4 CONTENT

Part 2 [MPEG4-2] and Part 3 [MPEG4-3] of the MPEG-4 standard (corresponding to Chapters 8 to 12 in this book) specify the coding tools to generate compressed audio and visual streams. Each semantic visual or audio object may be coded into one or more such streams. An MPEG-4 presentation that consists of

multiple such objects (and, hence, many streams) needs mechanisms to describe all the objects in the presentation. One part of this mechanism is embodied in the object descriptor framework [MPEG4-1, Clause 8]. All components of an MPEG-4 presentation are discovered recursively through object descriptors (ODs). The details of this approach are explained in this chapter.

Object description enumerates only the streams in a presentation and specifies how they relate to media objects. In order to assemble those media objects into a specific audiovisual scene, most MPEG-4 presentations also include a scene description. The scene description format, called *binary format for MPEG-4 scenes* (BIFS), is explained in detail in Chapter 4. In order to understand the present chapter, one must understand the general role of scene description and its conceptual difference from object description.

The scene description conveys the spatio-temporal layout of the media objects in the scene—that is, it captures the work of the creative content designers. Object description serves an asset management purpose; it allows locating streams and retrieving information about them. This information may be produced during authoring of the scene description, but—different from the scene description—it also may be subsequently amended or changed, for example, when media streams for a presentation are relocated to a different media server.

The basic concept of an OD is that of a container aggregating all the useful information about the corresponding object. This information is structured in a hierarchical manner through a set of subdescriptors. Each OD describes streams that are referred from a specific location in the scene description. In the case of audio streams, for example, this could be an `AudioSource` node (see Figure 3.1). The media streams that are related to this single `AudioSource` node constitute the *object* or *media object* that is described by the OD. More than one media stream can be associated with the object, as will be seen.

3.1.1 Syntactic Description Language

In order to document the syntax of the OD framework, the MPEG-4 Systems standard—as well as this chapter—uses the Syntactic Description Language (SDL). SDL is specified in [MPEG4-1, Clause 14]. This language is a convenient shorthand to express bitstream syntax of inherently object-oriented structures in a C++ kind of way. The strict syntax definition of SDL has the advantage that it enables the fully automatic generation of a code stub for the bitstream parser from the SDL code in [MPEG4-1].¹

A variable definition in SDL typically includes a directive to parse the value for this variable from a bitstream. Specifically, parsing `length` bits off

1. The other Parts of the MPEG-4 standard continue to use a less rigid pseudo-C notation for bitstream specification. This is mostly due to the fact that SDL has been developed in the same timeframe as the other Parts of the MPEG-4 standard.

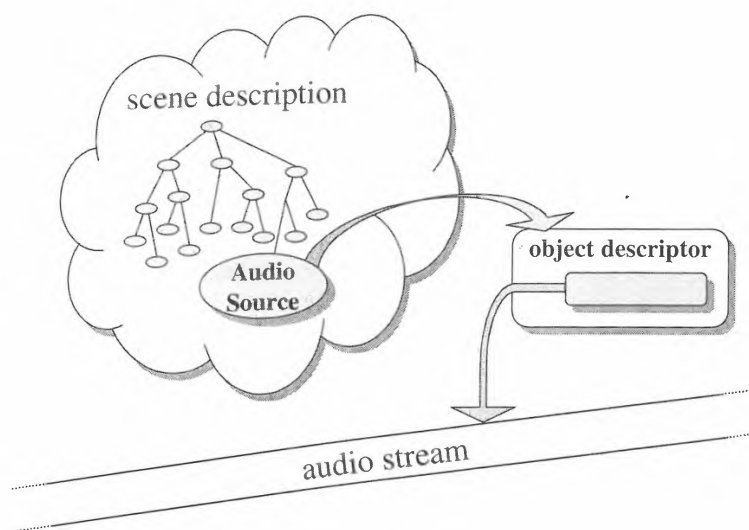


Fig. 3.1 BIFS node (AudioSource) pointing to a(n audio) stream through an OD.

the bitstream into a variable called `element_name` with a certain type is expressed as follows:

```
[aligned] type[(length)] element_name [= value];
```

where `type` is any of the data types—bit for arbitrary bit sequences, double for floating values, `int` for integer or unsigned `int` for unsigned integer. Square brackets express optional elements. For example, if `length` is missing, it means that the variable is not parsed from the bitstream but rather is a locally defined variable as in any programming language. The optional `aligned` attribute causes the reading of `length` bits to start from the next byte boundary, making the variable byte-aligned. If the variable `element_name` is assigned a `value`, this causes the parser to assert that the value read from the bitstream actually corresponds to this `value`. In the example,

```
aligned bit(32) picture_start_code=0x00000100;
```

the parser is asked to parse 32 bits from the next byte boundary and to check that this variable corresponds to the hexadecimal value 100.

Often, the number of bits for a certain syntax element is itself signaled in the bitstream. Therefore, it is permitted to use variables in the `length` field, as in the second line of the next example:

```
unsigned int(3) precision;
int(precision) DC;
```

A number of variables that constitute the data for an object may be arranged into classes:

```
class Foo {
    unsigned int(5) aVal;
```

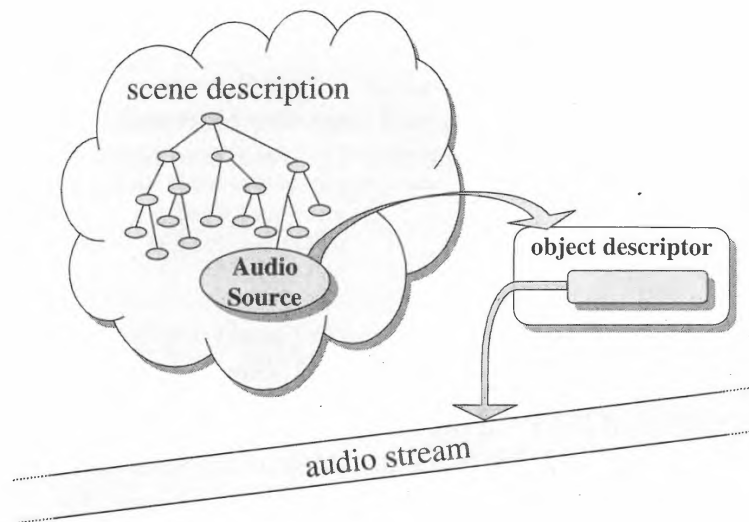


Fig. 3.1 BIFS node (AudioSource) pointing to a(n audio) stream through an OD.

the bitstream into a variable called `element_name` with a certain type is expressed as follows:

```
[aligned] type[(length)] element_name [= value];
```

where `type` is any of the data types—bit for arbitrary bit sequences, double for floating values, int for integer or unsigned int for unsigned integer. Square brackets express optional elements. For example, if `length` is missing, it means that the variable is not parsed from the bitstream but rather is a locally defined variable as in any programming language. The optional `aligned` attribute causes the reading of `length` bits to start from the next byte boundary, making the variable byte-aligned. If the variable `element_name` is assigned a value, this causes the parser to assert that the value read from the bitstream actually corresponds to this `value`. In the example,

```
aligned bit(32) picture_start_code=0x00000100;
```

the parser is asked to parse 32 bits from the next byte boundary and to check that this variable corresponds to the hexadecimal value 100.

Often, the number of bits for a certain syntax element is itself signaled in the bitstream. Therefore, it is permitted to use variables in the `length` field, as in the second line of the next example:

```
unsigned int(3) precision;
int(precision) DC;
```

A number of variables that constitute the data for an object may be arranged into classes:

```
class Foo {
    unsigned int(5) aVal;
```

```

        bit(3) bVal;
    }

```

Each time an instance of class `Foo` occurs in another class, a five-bit value `aVal` and a three-bit value `bVal` are parsed from the bitstream. For example, class `Bar` will parse five elements of type `Foo` (i.e., read five bytes from the bitstream) with the following code:

```

class Bar {
    Foo aFoo[5];
}

```

The array elements read into `aFoo` can be further used in the SDL code of class `Bar` with the familiar syntax `aFoo[i].aVal` and `aFoo[i].bVal`, yielding the `aVal` and `bVal` values of the i^{th} array element.

After those fairly simple definitions, it comes as no surprise that classes may inherit from other classes, as in the next example:

```

class FooBar extends Foo {
    int(8) cVal;
}

```

`FooBar` now actually has the elements `aVal`, `bVal`, and `cVal`. In other words, parsing of an instance of class `FooBar`, as in the next example,

```

class ParseAFooBar {
    FooBar aFooBar;
}

```

results in reading two bytes (5 + 3 + 8 bit) that are accessible as `aFooBar.aVal`, `aFooBar.bVal`, and `aFooBar.cVal`.

Furthermore, classes can be identified with a tag value (0x23 in the following example) that is parsed from the bitstream before reading any element within the class:

```

class ClassWithID : aligned bit(8) classTag=0x23 {
    FooBar aFooBar;
}

```

So, if an instance of `ClassWithID` is expected, as in

```

class ParseAClassWithID {
    int (8) a
    ClassWithID aCwID;
}

```

an eight-bit value other than 0x23 is read from the bitstream after reading the eight bits of `a`, and thus the parser knows that an error has occurred. Although this may be useful to know, the major use of tags is the discrimination of different classes that may occur in the bitstream at the same location. Those classes would have to derive from a common base class:

```

class ClassesWithID : aligned bit(8) classTag=0x23 .. 0x33 {
    FooBar aFooBar;
}

```

in a manner like:

```
class FirstClass extends ClassesWithID : aligned bit(8) classTag=0x23
{
    int(8) aaValue;
}
class SecondClass extends ClassesWithID : aligned bit(8)
{
    classTag=0x24 {
        int(8) bbValue;
    }
}
```

giving unique class tags to FirstClass and SecondClass in the range of tags specified for classes that inherit from ClassesWithID.

Now, instances of the derived classes of ClassesWithID can be parsed through

```
class ParseClassesWithID {
    int (8) a
    ClassesWithID aCwID;
}
```

If a value of 0x23 is read after the eight bits of a, an instance of FirstClass will be read; if it is 0x24, it will be an instance of SecondClass; in all other cases with tag values between 0x25 and 0x33, an instance of the base class ClassesWithID will be read.

Base classes also can be used to organize a set of classes and provide a common name scope for their tag values. For this purpose, similar to C++, abstract classes can be defined that cannot be instantiated (i.e., they cannot be read from the bitstream) and which do not have tag values of their own. The derived classes can then populate the full range of tag values, as given by the type declaration of the tag. So, in the following example,

```
abstract class MyClassesWithID : aligned bit(8) classTag=0 {
    Foobar aFoobar;
}
class FirstClass extends MyClassesWithID : aligned bit(8)
{
    classTag=0x23 {
        int(8) aaValue;
    }
}
class SecondClass extends MyClassesWithID : aligned bit(8)
{
    classTag=0x24 {
        int(8) bbValue;
    }
}
class OtherClass : aligned bit(8) classTag=0x24 {
    int(8) ccValue;
}
class ParseClassesWithID {
    int (8) a;
    OtherClass aClass;
    MyClassesWithID aCwID;
}
```

OtherClass may reuse the tag value of 0×24 , as it is not derived from `ClassesWithID`. `ParseClassesWithID` parses an integer variable `a` and one instance of `OtherClass` before parsing an instance of `MyClassesWithID`. Here, the parser has to encounter either a value of 0×23 or 0×24 to identify a `FirstClass` or `SecondClass` instance. Otherwise, the parser would assume it has encountered an unknown derived class from `ClassesWithID`. Nevertheless, parsing would have to stop here with an error, because parsing of that unknown class is not possible.

Therefore, a special type of classes exists that is both expandable and can be skipped easily, as the overall length of this class is encoded. The length indication is conveyed immediately after the tag (if any) and before any parsable variable.

```
abstract expandable (228-1) class ClassesWithID : aligned bit(8)
    classTag=0 {
        Foobar aFoobar;
    }
```

The derived classes of this version of `ClassesWithID` would have a length indication following the class tag, limiting the class size to $2^{28}-1$. When an instance of such a derived class is encountered with a length bigger than what can be computed from the class definition, the parser knows that some trailing data are conveyed in that class which, possibly, it cannot parse but can ignore and continue parsing after skipping that data. When using expandable classes, the parser does not have to fail when parsing an unknown derived class out of a number of derived classes from a base class. The length indication is helpful here to skip such a class instance.

These basic² SDL principles outlined above are used in the subsequent sections to introduce the basic syntactic structures of the OD framework.

3.1.2 Object Description

The syntax specification of all descriptors [MPEG4-1, Clause 8.6] is derived from a common base class, called `BaseDescriptor` (see Table 3.1). It specifies that descriptors always are identified by a numeric tag of eight-bit length. Furthermore, it limits the length of descriptors to $2^{28}-1$ bytes and, through the use of an expandable class definition, specifies that each derived descriptor

Table 3.1 Base Descriptor: The template for other descriptors

```
abstract aligned(8) expandable(228-1) class BaseDescriptor : bit(8) tag=0 {
    // empty. To be filled by classes extending this class.
}
```

2. Features of SDL that are not needed for the code fragments given in this book have been omitted.

defined in the MPEG-4 specification can be extended in the future by syntax elements that are added at the end of the descriptor.

The syntax of the OD [MPEG4-1, Clause 8.6.3] presented in Table 3.2 shows that `ObjectDescriptor` inherits from `ObjectDescriptorBase` which in turn inherits from `BaseDescriptor` (Table 3.1). The latter is another empty abstract base class for different types of ODs.

For labeling, each OD has an `ObjectDescriptorID`. This ID allows the unique identification of a set of streams in an MPEG-4 presentation. Additionally, it allows the scene description stream, BIFS, to refer to a specific descriptor.

An OD may further contain either only a URL or an actual OD. If it includes only a URL, it means that the actual OD can be found at the location pointed to by the URL. This is useful for managing large assets of media, in which not all media and their descriptions are located on a single media server. Basically, URLs would allow assembling one MPEG-4 presentation from content distributed all over the globe in a manner comparable to today's compound Web pages, which assemble elements from multiple Web servers.

The content of the OD (in the absence of a URL) is a series of subdescriptors. These subdescriptors describe four elements:

- ☞ Individual elementary streams (ESs)
- ☞ Semantic information about an object (OCI³ descriptor)
- ☞ Hooks for content access management (IPMP⁴ descriptor pointer)
- ☞ A placeholder for future extension descriptors

ES description is detailed first, with the OCI and IPMP frameworks coming later in the section about semantic description and access management.

Table 3.2 OD syntax

```

class ObjectDescriptor extends ObjectDescriptorBase : bit(8)
{
    tag=ObjectDescrTag {
        bit(10) ObjectDescriptorID;
        bit(1) URL_Flag;
        const bit(5) reserved=0b1111.1;
        if (URL_Flag) {
            bit(8) URLlength;
            bit(8) URLstring[URLlength];
        } else {
            ES_Descriptor esDescr[1 .. 255];
            OCI_Descriptor ociDescr[0 .. 255];
            IPMP_DescriptorPointer ipmpDescrPtr[0 .. 255];
        }
        ExtensionDescriptor extDescr[0 .. 255];
    }
}

```

3. Object content information.

4. Intellectual property management and protection.

3.1.3 Stream Description

Each individual stream—or, more precisely, each *elementary stream*—is described by an elementary stream descriptor (ES descriptor or ESD; see Table 3.3) specified in [MPEG4-1, Clause 8.6.5]. An ES is conceived as a flow of (possibly compressed) data that originates from a single source in the sender and terminates in a single sink at the receiver. In a simple case, a media encoder (audio or video) and the corresponding media decoder are communicating through one ES (media encoder/decoder 1 in Figure 3.2). However, a single media decoder also may consume multiple ESs concurrently, specifically in case of a scalable encoding (media encoder/decoder 2 in Figure 3.2). Formally, a set of one or more ESs that are jointly decoded to recover the media object makes up a *media stream* in the MPEG-4 context.

Each ESD provides detailed information about the associated stream, including its location, type, encoding, bit rate, buffer requirements, and decoder specific information. As a comparison, only a subset of this information could be conveyed with multipurpose Internet mail extensions (MIME) [RFC2045] or the session description protocol (SDP) [RFC2327], mechanisms often used in Internet streaming applications for stream description.

Looking at the ESD elements in detail, stream location is provided first. This is done through one of two mechanisms: It can be done solely by means of an elementary stream ID (ES_ID). In that case, it is assumed that the media

Table 3.3 ESD syntax

```

class ES_Descriptor extends BaseDescriptor : bit(8) tag=ES_DescrTag {
    bit(16) ES_ID;
    bit(1) streamDependenceFlag;
    bit(1) URL_Flag;
    bit(1) OCRstreamFlag;
    bit(5) streamPriority;
    if (streamDependenceFlag)
        bit(16) dependsOn_ES_ID;
    if (URL_Flag) {
        bit(8) URLlength;
        bit(8) URLstring[URLlength];
    }
    if (OCRstreamFlag)
        bit(16) OCR_ES_Id;
    DecoderConfigDescriptor decConfigDescr;
    SLConfigDescriptor slConfigDescr;
    IPI_DescrPointer ipiPtr[0 .. 1];
    IP_IdentificationDataSet ipIDS[0 .. 255];
    IPMP_DescriptorPointer ipmpDescrPtr[0 .. 255];
    LanguageDescriptor langDescr[0 .. 255];
    QoS_Descriptor qosDescr[0 .. 1];
    RegistrationDescriptor regDescr[0 .. 1];
    ExtensionDescriptor extDescr[0 .. 255];
}

```

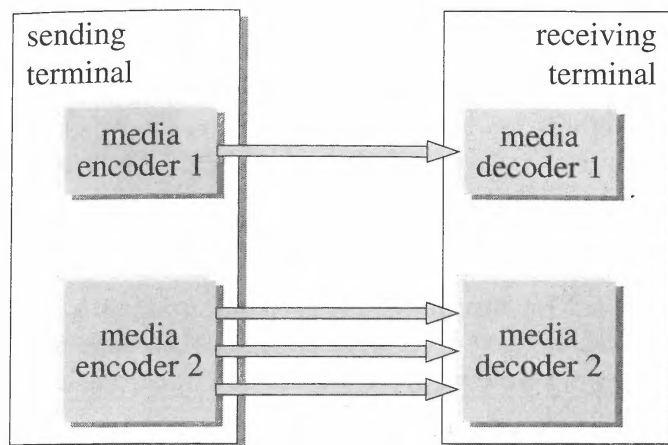


Fig. 3.2 Media encoder and decoder can be connected by one (case 1) or more (case 2) elementary streams.

server storing this stream will be able to connect this `ES_ID` to some stream resource in an application-specific way. Second, the URL field can be used. This allows locating ESs in a more versatile manner. In both cases, however, the actual description of the stream is done by the ESD at hand.

The flags after `ES_ID` signal whether the related syntax elements are present in the ES descriptor in question. The `streamPriority` element signals the relative importance of the streams within a presentation: The higher the value, the more important the stream. This information is mainly of use for a stream server or a gateway that can evaluate the stream priority values for all streams in a presentation and then group them into different transmission channels accordingly. In other words, the semantics of this field must be determined within a specific application scope.

As mentioned before, ODs and ESDs are containers for information about the associated object and ES, respectively. Therefore, the subsequent syntax elements relate to a number of different features that are introduced in the sections to come. First, the stream dependence mechanism defines how streams hierarchically depend on each other by pointing to the `ES_ID` of the stream upon which this one directly depends. The usage will be further discussed in the stream-dependence section. The `OCRstreamFlag` and `OCR_ES_ID` allow tying streams to a specific time base by pointing to the `ES_ID` of the stream that conveys this time base information. The usage will be discussed in the section on synchronization.

The remainder of the ESD is a series of subdescriptors with more specific and optional information. Details on `IPI_DescrPointer`, `IP_IdentificationDataSet`, `IPMP_DescriptorPointer`, and language descriptors are given in the IPMP and OCI sections. `SLConfigDescriptor` describes synchronization-related properties and is explained in the synchronization section. The remaining subdescriptors are covered in the following sections.

3.1.3.1 Decoder Configuration The `DecoderConfigDescriptor` (Table 3.4) provides setup information required by the decoder to process the stream in question. First, the stream is roughly classified through the `streamType` value as visual, audio, scene description, and so on. Then, the `object-TypeIndication` gives further information on the actual coding employed, such as JPEG, MPEG-1, MPEG-2, and MPEG-4. For legacy stream types (MPEG-1, MPEG-2), profile and level information is directly included, as applicable.

The `upStream` flag is set for streams that provide upstream information from the client terminal back to the server. Attaching such a stream to an OD, in addition to a regular stream flowing from server to client, means that this object has its own back channel.

The remaining values in `DecoderConfigDescriptor` indicate the buffer size required for the decoding buffer as well as the maximum and the average bit rates that should be expected when receiving the stream.

The `DecoderSpecificInfo` descriptor serves as an opaque container from the MPEG-4 Systems point of view and carries information about the decoder configuration needed to decode the stream properly.

Finally, the `profileLevelIndicationIndexDescriptor` allows associating this stream to one or more complexity categories specified for the overall MPEG-4 presentation, as detailed in the complexity section below.

3.1.3.2 Quality of Service Descriptor Comparable to the `streamPriority` element in the ESD, the quality of service (QoS) descriptor is mostly of use for the media server and for gateways that intelligently forward streams or sets of streams. The QoS descriptor consists of either an (application-dependent) index of a predefined QoS scenario or a set of QoS qualifiers (Table 3.5). QoS qualifiers are defined as derived classes from the abstract `QoS_Qualifier` class.

Table 3.4 Decoder Configuration Descriptor syntax

```
class DecoderConfigDescriptor extends BaseDescriptor : bit(8)
{
    tag=DecoderConfigDescrTag {
        bit(8) objectTypeIndication;
        bit(6) streamType;
        bit(1) upStream;
        const bit(1) reserved=1;
        bit(24) bufferSizeDB;
        bit(32) maxBitrate;
        bit(32) avgBitrate;
        DecoderSpecificInfo decSpecificInfo[0 .. 1];
        profileLevelIndicationIndexDescriptor profileLevelIndicationIndexDescr
            [0..255];
    }
}
```

Table 3.5 QoS Descriptor syntax and QoS Qualifier definition

```

class QoS_Descriptor extends BaseDescriptor : bit(8) tag=QoS_DescrTag {
    bit(8) predefined;
    if (predefined==0) {
        QoS_Qualifier qualifiers[];
    }
}

abstract aligned(8) expandable(228-1) class QoS_Qualifier : bit(8)
    tag=0x01..0xff {
    // empty. To be filled by classes extending this class.
}

```

A number of QoS qualifiers have been predefined to express the following:

- ☞ Maximum end-to-end delay for the stream in microseconds
- ☞ Preferred end-to-end delay for the stream in microseconds
- ☞ Allowable loss probability of any single AU as a fractional value between 0.0 and 1.0
- ☞ Maximum allowable number of consecutively lost AUs
- ☞ Maximum size of an AU in bytes
- ☞ Average size of an AU in bytes
- ☞ Maximum arrival rate of AUs in AU/second

This set of qualifiers may need application-specific extension, depending on the selection of a specific QoS framework.

3.1.3.3 Registration Descriptor Finally, a registration descriptor (Table 3.6) is provided to enable MPEG-4 users to unambiguously carry non-MPEG-4 ESs. Although MPEG does not (prefer to) open up the set of allowed ESs, it is acknowledged that applications will still have the need to convey such private streams. The registration descriptor provides a method to unambiguously identify the characteristics of these streams.

As in other such cases, ISO will issue a Call for Registration Authority, a body that will subsequently manage the space of `formatIdentifier` values and make sure that they are uniquely assigned. The second field in the descriptor may hold any additional information, as requested by the owner of a certain `formatIdentifier`.

Table 3.6 Registration Descriptor syntax

```

class RegistrationDescriptor extends BaseDescriptor : bit(8)
    tag=RegistrationDescrTag {
    bit(32) formatIdentifier;
    bit(8) additionalIdentificationInfo[sizeofInstance-4];
}

```

3.1.4 Stream Relationship Description

The mechanisms to describe a single ES for a media object have been introduced. Describing an object with a single ES description consists in creating an ESD and placing it within an OD. However, the OD framework has some more features that are specified in [MPEG4-1, Clause 8.7]. For example, it is possible to describe various alternative representations of the same media object.

In a simple case, a single media object is encoded in different qualities. The corresponding media streams can be generated independently, including for each of them an ESD with the necessary parameters, such as coding format and bit rate. Those ESDs are simply put together in one single OD. The client terminal will then select one of the streams and request it from the server for playback (see Figure 3.3).

In some application contexts, scalable coding is more adequate than providing a set of independent quality streams—for example, in the case of multicast distribution, in which the client cannot request a specific quality from the server. A base quality (base layer) is provided by a first ES that can be decoded independently. Depending on the concrete scalability algorithms used by the media codec, this quality can be improved, possibly in multiple steps, corresponding to the multiple ESs (enhancement layers) that are generated. Here it is important to know in which order to pass the set of streams to the media decoder (see Figure 3.4). This is achieved with the stream-dependency mechanism of the ESD. Each stream points to the one it depends on; more precisely, it points to the `ES_ID` of that stream using the `dependsOn_ES_ID` syntax element. The client terminal in that case will select one or more of these streams for playback, starting, of course, from the base layer stream.

Both in the case of different quality alternatives for a media object and in the case of scalable coding, the receiving terminal has a choice. The terminal needs to figure out which of the streams to use, depending on its capabilities. The terminal can do this by simply evaluating all the ODs and using custom

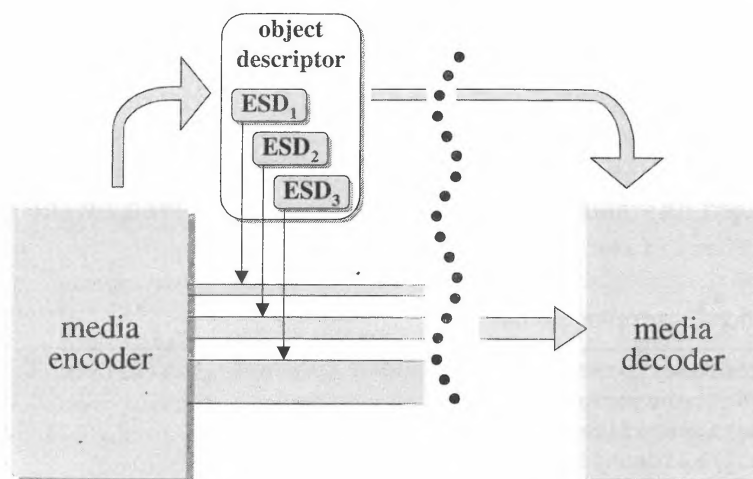


Fig. 3.3 Selecting one of a set of simulcast media streams.

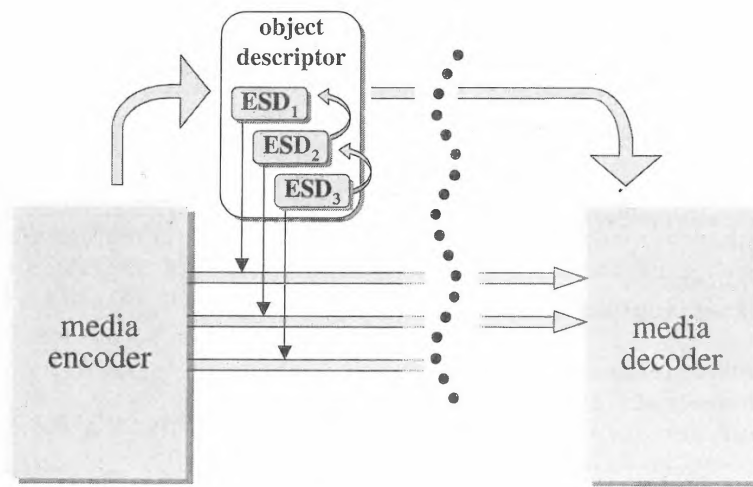


Fig. 3.4 Selecting a set of scalable streams.

rules to decide what to do. More likely, however, profile and level definitions are used for this purpose, as will be seen in the next section.

3.1.5 Content Complexity Description

The question of content complexity is important and has to be answered up front before an MPEG-4 terminal can decide whether it will attempt to decode and present that content at all. For a single media stream, it may be simple to examine the OD for that stream in all its advertised variations, and then to decide which one the terminal is capable of presenting. But now assume that the content consists of a large number of media objects embedded into a large scene. In that case, an examination of all related ODs would be quite arduous. Because of that, another mechanism has been implemented to answer the question of content complexity. It is basically a set of labels indicating profiles and levels for the content. The profile and level concepts will be addressed in detail in Chapter 13.

The profile and level indications are conveyed by means of two types of descriptors: a derivative of the OD called the *initial object descriptor* (IOD, Table 3.7), and the *extension profile level descriptor* (Table 3.8).

The IOD is identical to the regular OD, except that it features the said elements for signaling the different profiling dimensions for scene graph, audio, visual, graphics, and OD (see Chapter 13). Additionally, a flag (`includeInlineProfileLevelFlag`) signals whether these profiles and levels correspond to the sum of the complexity of the overall scene including all sub-scenes⁵ (`includeInlineProfileLevelFlag=1`), or whether those complexities

5. Scene descriptions can be recursively included in each other using the `Inline` node, which will be introduced in Chapter 4.

Table 3.7 Initial OD syntax

```

class InitialObjectDescriptor extends ObjectDescriptorBase : bit(8)
    tag=InitialObjectDescrTag {
        bit(10) ObjectDescriptorID;
        bit(1) URL_Flag;
        bit(1) includeInlineProfileLevelFlag;
        const bit(4) reserved=0b1111;
        if (URL_Flag) {
            bit(8) URLlength;
            bit(8) URLstring[URLlength];
        } else {
            bit(8) ODPprofileLevelIndication;
            bit(8) sceneProfileLevelIndication;
            bit(8) audioProfileLevelIndication;
            bit(8) visualProfileLevelIndication;
            bit(8) graphicsProfileLevelIndication;
            ES_Descriptor esDescr[1 .. 255];
            OCI_Descriptor ociDescr[0 .. 255];
            IPMP_DescriptorPointer ipmpDescrPtr[0 .. 255];
        }
        ExtensionDescriptor extDescr[0 .. 255];
    }

```

Table 3.8 Extension Profile Level Descriptor syntax

```

class ExtensionProfileLevelDescriptor() extends BaseDescriptor :
    bit(8) ExtensionProfileLevelDescrTag {
        bit(8) profileLevelIndicationIndex;
        bit(8) ODPprofileLevelIndication;
        bit(8) sceneProfileLevelIndication;
        bit(8) audioProfileLevelIndication;
        bit(8) visualProfileLevelIndication;
        bit(8) graphicsProfileLevelIndication;
        bit(8) MPEGJProfileLevelIndication;
    }

```

are signaled by their own InitialObjectDescriptors or ExtensionProfileLevelDescriptors.

The ExtensionProfileLevelDescriptor is necessary because a single complexity indication for an MPEG-4 presentation may not be sufficient. As explained, ODs allow aggregating multiple representations of the same content with different characteristics. Therefore, a signaling means for the complexity of these different representations in terms of profiles and levels is required. This is achieved by attaching multiple ExtensionProfileLevelDescriptors to the initial OD, each describing one of these representations and uniquely labeled by a profileLevelIndicationIndex. Additionally, this descriptor serves to signal MPEG-J profiles (see Chapter 5) that did not exist in Version 1 of the MPEG-4 Systems standard.

A stream that belongs to a given representation (or complexity category) may use the `profileLevelIndicationIndexDescriptor` (Table 3.9) from its `DecoderConfigDescriptor` to reference the appropriate `profileLevelIndicationIndex`.

Table 3.9 Profile Level Indication Index Descriptor syntax

```
class ProfileLevelIndicationIndexDescriptor () extends BaseDescriptor
:   bit(8) ProfileLevelIndicationIndexDescrTag {
    bit(8) profileLevelIndicationIndex;
}
```

3.1.6 Streaming ODs

The previous sections explained how media objects and media streams are defined and described through ODs and ESDs. They also detailed how to describe different encoded versions of the same media object and how to indicate the complexity of these objects. The next step is to communicate the ODs with all this information from the sender to the client terminal.

In MPEG-4 the focus is on enabling dynamic multimedia content. This led to a design in which ODs can be sent, updated, or deleted at any time during an MPEG-4 presentation. In fact, ODs are sent within their own OD stream [MPEG4-1, Clause 8.5]; that is, they are generally processed in the same way as other media streams or the scene description streams.

This MPEG-4 approach is quite different from existing mechanisms for signaling stream properties. Both SDP [RFC2327], in conjunction with the real-time streaming protocol (RTSP) [RFC2326] used in some Internet streaming applications, and the ITU-T Recommendation H.245 [H245] used for videotelephone and videoconferencing applications perform the signaling of stream properties only at the beginning of the multimedia presentation or communication.

Within the OD stream, a set of OD commands can be conveyed:

- ☞ OD update
- ☞ OD remove
- ☞ ESD update
- ☞ ESD remove
- ☞ OD execute

The OD update command conveys a set of ODs or initial ODs. Adding and updating an OD are treated the same way. For each updated OD, the terminal will check whether existing streams were removed from it or new streams have been added. Through an OD update, all the auxiliary information related to OCI and IPMP can be updated, too.

The ESD update makes it possible to convey individual ESDs and, therefore, plug an ES into an OD after that OD is already conveyed to the receiving

terminal. It advertises at any stage during a presentation that an additional version of a media object just became available—for example, an enhanced quality version. Removing ESDs or ODs indicates future unavailability of the associated streams.

The OD execute command exists in order to let the terminal know upfront for which streams a decoder needs to be set up immediately. In the absence of such OD execute commands, the terminal may set up decoders only when the scene description actually requests the playback of some media stream. In that case, an extra delay for setting up the decoder must be expected, which will be implementation-dependent. The need for the OD execute command was established very late in the standardization process. Therefore it is found only in the second edition of the MPEG-4 Systems standard [MPEG4-1].

All these commands follow a syntax similar to that of the descriptors encountered before. They are given in Table 3.10 for completeness. All commands derive from the abstract class `BaseCommand`. As mentioned earlier, an abstract class defines a name scope for the class tags. So, `BaseCommand` defines

Table 3.10 Syntax of OD commands

```

abstract aligned(8) expandable(228-1) class BaseDescriptor : bit(8) tag=0 {
    // empty. To be filled by classes extending this class.
}

class ObjectDescriptorUpdate extends BaseCommand : bit(8)
    tag=ObjectDescrUpdateTag {
        ObjectDescriptorBase OD[1 .. 255];
    }

class ObjectDescriptorRemove extends BaseCommand : bit(8)
    tag=ObjectDescrRemoveTag {
        bit(10) objectDescriptorId[(sizeofInstance*8)/10];
    }

class ES_DescriptorUpdate extends BaseCommand : bit(8)
    tag=ES_DescrUpdateTag {
        bit(10) objectDescriptorId;
        ES_Descriptor esDescr[1 .. 255];
    }

class ES_DescriptorRemove extends BaseCommand : bit(8)
    tag=ES_DescrRemoveTag {
        bit(10) objectDescriptorId;
        aligned (8) bit(16) ES_ID[1..255];
    }

class ObjectDescriptorExecute extends BaseCommand : bit(8) tag=
    ObjectDescriptorExecuteTag {
        bit(10) objectDescriptorId[(sizeofInstance*8)/10];
    }

```

a new name scope for the command tags that is distinct from the name scope for descriptor tags.

3.1.6.1 Streaming Configuration Information in ODs In previous standards (such as MPEG-2), decoder configuration information (like coding format and coding options) is repeated regularly as part of the media ES itself in order to allow the random access to a stream, as indicated in Figure 3.5a. In case of MPEG-2, some of this configuration information is additionally repeated in the Systems layer [MPEG2-1]. In MPEG-4, it is suggested to convey the decoder configuration information within the `DecoderSpecificInfo` element (Table 3.4), avoiding any duplication of this information in the ES itself.

This approach makes it possible to adapt the delivery rules for the decoder configuration information to the delivery medium. In case of unicast delivery without the ability to randomly tune in, decoder configuration information is delivered up front and no repetition is needed. In case of a broadcast medium, the ODs can easily be repeated as often as needed within their OD stream, as indicated in Figure 3.5b. The arrows in this principal drawing indicate that it is possible to maintain a temporal link between the OD stream and the media streams through the use of time stamps.

Unfortunately, this ideal approach cannot always be followed, because the MPEG-4 toolbox approach allows the use of MPEG-4 Visual or Audio coding

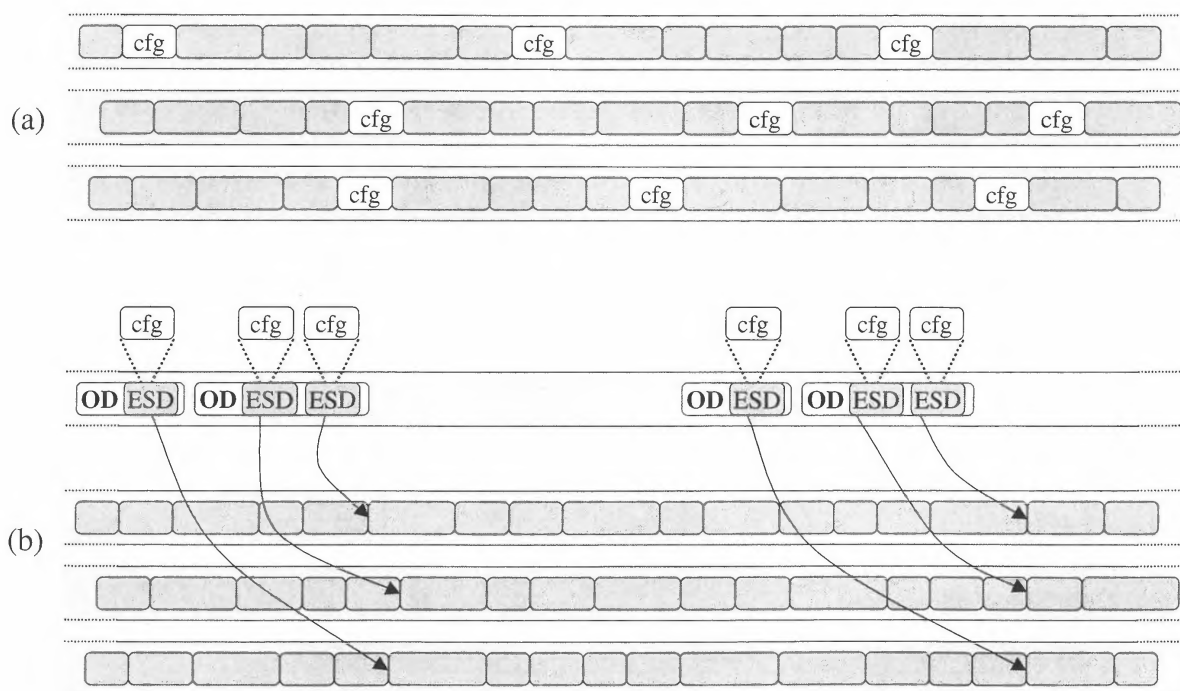


Fig. 3.5 Carriage of configuration (cfg) information (a) in each ES (possibly repeated), as in MPEG-2, and (b) in the decoder-specific info descriptor within a separate OD stream, as in MPEG-4.

tools independently of the Systems tools, which include the ODs. In the absence of ODs, the ES syntax has to revert to the approach shown in Figure 3.5a.

3.1.7 Linking a Scene to Its Media Streams

It was mentioned before (Figure 3.1 and Figure 3.2) that ODs are the link between an individual media object as represented in the scene description and its associated media stream(s). Given that both ODs and the scene description are conveyed in streams, it becomes apparent that further rules are needed to associate an entire scene description stream to an OD stream [MPEG4-1, Clause 8.7.2].

3.1.7.1 Association of OD Streams and Scene Description Streams OD streams are usually associated with a scene description (BIFS) stream, as that stream conveys the information on how to assemble the various media streams described within the OD stream into an MPEG-4 presentation. The means to associate OD and BIFS streams is another OD. The ESDs for an OD stream and its associated scene description stream are put into one OD or IOD and the scene description stream is set to depend on its OD stream (Figure 3.6).

It is also possible to split the scene description information into more than one stream—basically, to do a scalable encoding of the scene description. The order of relevance of the various BIFS streams is again signaled through the stream dependency mechanism in the OD (Figure 3.7), as done for scalable media streams.

Splitting a single BIFS stream in this way is notably different than partitioning a scene in subscenes through `Inline` nodes (see Chapter 4). It just means that some of the scene description data (called BIFS commands) that construct a scene are sent in one stream and some in the other(s). The BIFS decoder in the receiving terminal merges the BIFS commands from those streams in the right temporal order prior to decoding. It is up to the authoring tool to use this feature in a meaningful way, for example to support error resilience for the scene description data.

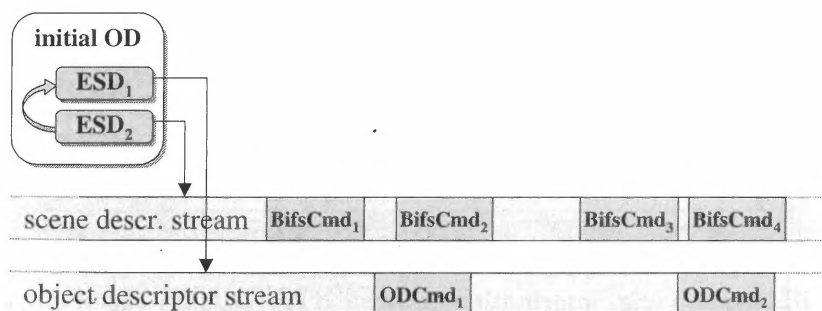


Fig. 3.6 Associating a scene description stream with its OD stream.

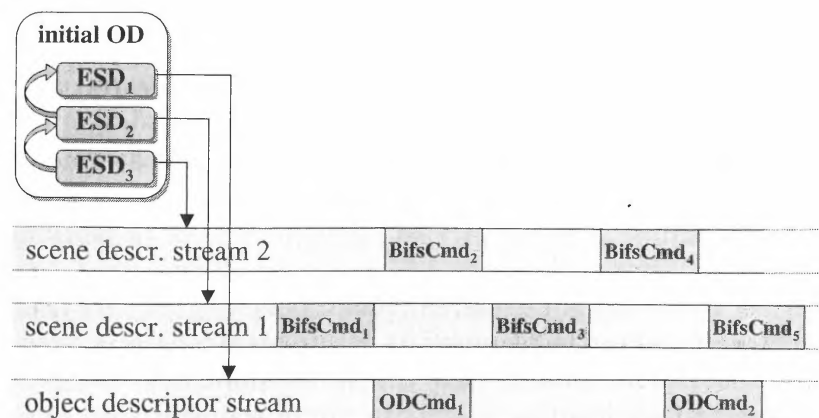


Fig. 3.7 Scalable scene description.

3.1.7.2 Association of MPEG-4 Presentations Another real-world question is how to author small MPEG-4 presentations and plug them together into bigger presentations later on. As an example, consider a main audiovisual show that is presented together with an overlaid ad banner including animated graphics and its own stamp-sized video stream (see Figure 3.8).

The authors for these two items are likely to work independently, but the overall presentation still needs to be assembled from both parts. Notably, in

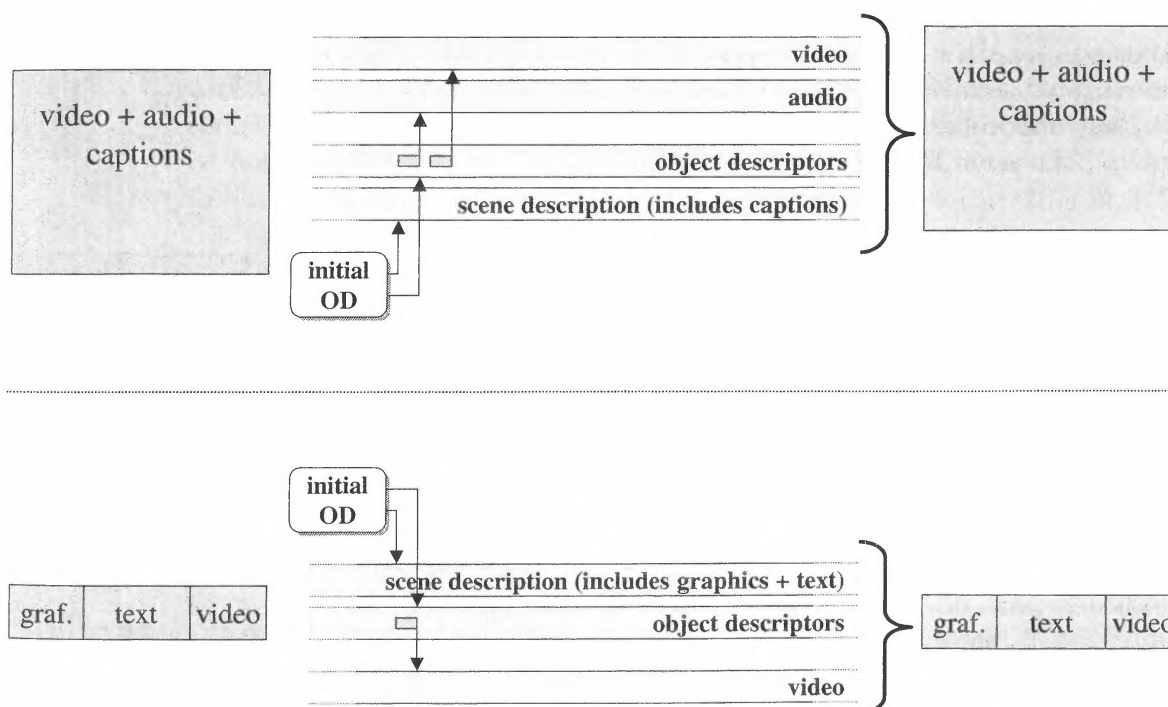


Fig. 3.8 Authoring two independent scenes: a main audiovisual program (top) and a banner-ad-like portion consisting of graphics, text, and video elements (bottom).

MPEG-4 this must be possible without reediting either content, because in fact the assembly will take place at the receiver. Unfortunately, independent editing means that both partial presentations may have used the same numerical identifiers to mean different things.

The proper solution for such problems is the definition of name scopes for the various parts of the content. This means, similar to the name scopes for descriptor tags encountered before, those identifiers have to be unique only within a well-defined scope.

BIFS defines a mechanism to establish subscenes, called the `Inline` node (as will be further explained in Chapter 4). Each such subscene constitutes a separate name scope. This means that the same identifier values for BIFS nodes can be used in each subscene. This name scoping rule needs to be extended to the references from BIFS to OD and from OD to media streams, which are usually made through `OD_ID` and `ES_ID`.

Consequently, name scopes in the OD domain are defined the following way: Each OD that points to BIFS and OD streams opens a new name scope. In fact such an OD will be either the initial OD of a scene or an OD that is pointed to by an `Inline` node. So this translates back to the BIFS rule that each `Inline` node opens a new name scope.

For the example in Figure 3.9, this means that `OD_IDS` and `ES_IDS`, as

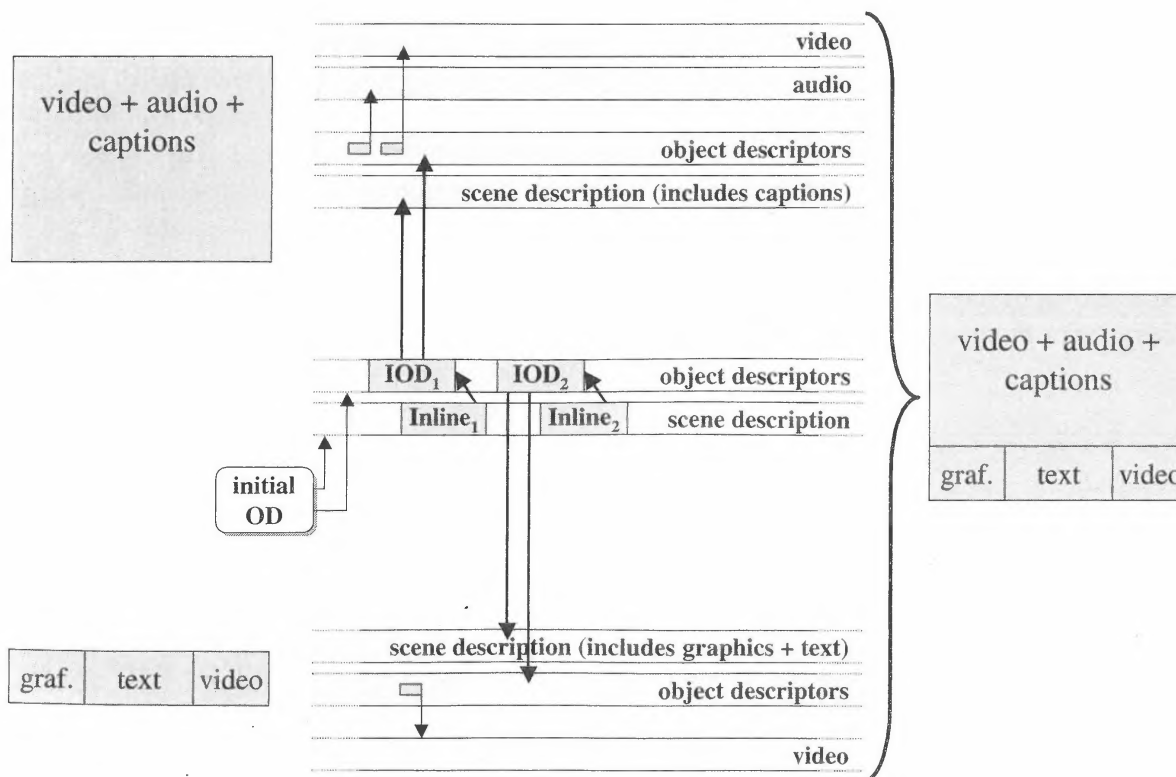


Fig. 3.9 Assembling a scene from two subscenes.

well as nodeIDs, routeIDs, and so on in BIFS, can be assigned independently in both individual scenes. The initial ODs (see Figure 3.8) for both scenes open one unique name scope for each scene. In these conditions, there will be no conflict when the scenes are composed in a combined presentation.

The process to compose the two scenes into one is the following: Another piece of scene description has to be authored that *inlines* the other two scenes as subscenes, defining the right place on the screen for each of them, of course. The initial ODs of the two subscenes will become part of the OD stream for the composite scene, as indicated in Figure 3.9. Starting from the new initial OD, the receiver will be able to reconstruct the composite scene.

3.1.8 MPEG-4 Content Access Procedure

In the first part of this chapter, all the elements necessary to access MPEG-4 content have been introduced. Content access always starts with an initial OD (see the top left of Figure 3.10). The IOD contains pointers to at least the two

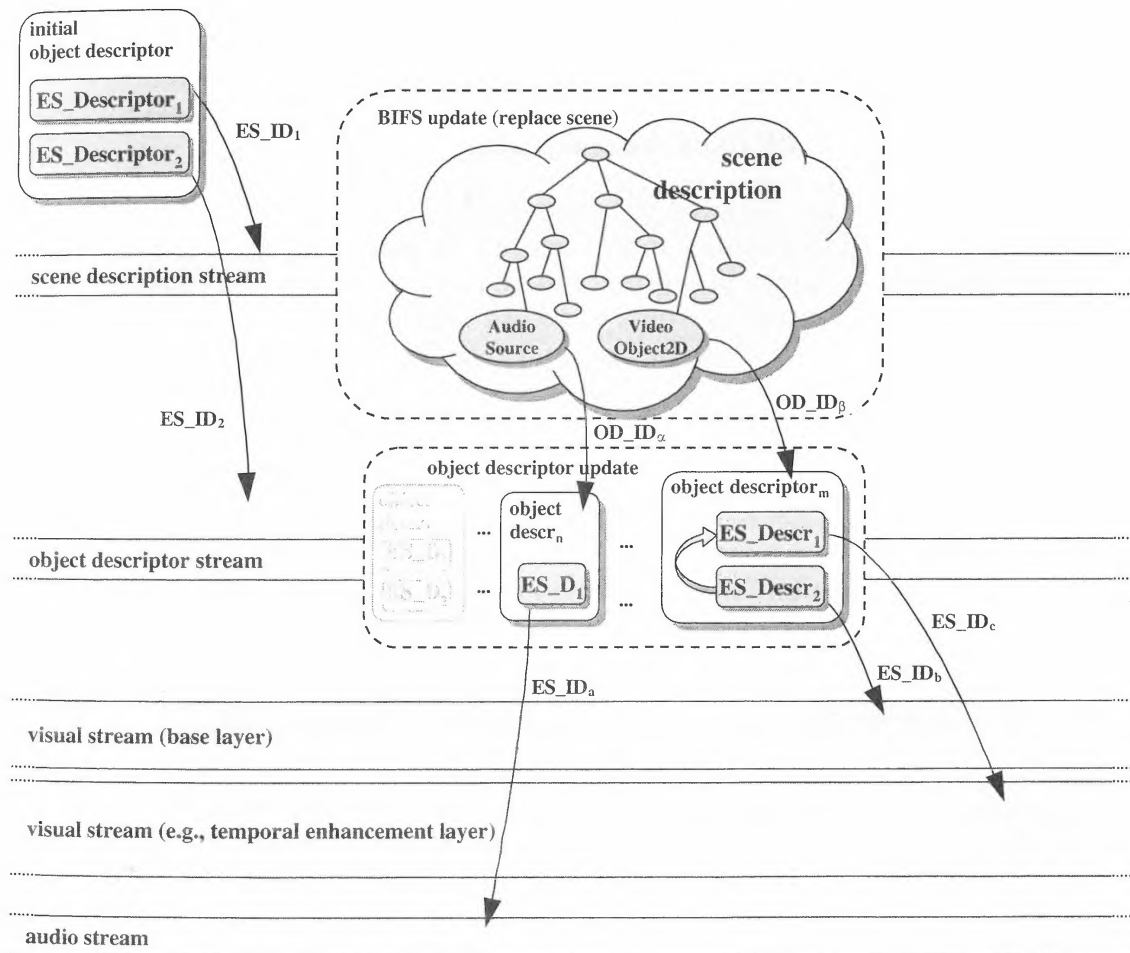


Fig. 3.10 MPEG-4 content access procedure.

essential streams: a scene description (BIFS) stream and an OD stream. The ODs in the OD stream allow the location of the additional media streams used in the scene and have all the information needed to initialize the corresponding media decoders. The streams are then referenced from the BIFS scene through pointers to their respective ODs.

This content access procedure [MPEG4-1, Clause 8.7.3] has been formalized in the MPEG-4 standard as a series of calls to an API, called the DMIF Application Interface (DAI). This API is specified in Part 6 of the MPEG-4 standard, *Delivery Multimedia Integration Framework* [MPEG4-6]. The DAI is an abstraction layer that enables the description of the content access procedure independently of the actual transport and signaling protocols used.

MPEG-4 does not define the transport or delivery of its coded data. Therefore, depending on the concrete MPEG-4 application scenario, different means may exist to communicate the IOD to the receiver. For example, this could be achieved by publishing URLs for IODs on Web pages, by using a suitable signaling protocol such as RTSP [RFC2326] or ITU-T H.245 [H245], or by embedding the IOD in an MPEG-2 transport stream [MPEG2-1]. Some of those mechanisms will be further explored in Chapter 7.

3.2 SEMANTIC DESCRIPTION AND ACCESS MANAGEMENT

The previous section detailed the essentials of MPEG-4 stream description necessary to enable simple content access. In addition, there are auxiliary streams and descriptors providing a little more semantic information about media objects and enabling content access management in order to prevent unauthorized content distribution and use. Two major types of auxiliary streams are defined in MPEG-4 Systems—OCI streams and IPMP streams. The intent of OCI is clear: It provides what is called *metadata*, describing objects semantically. IPMP provides hooks to control the legitimate usage of MPEG-4 content.

3.2.1 Object Content Information: Meta Information About Objects

A set of descriptors and a stream type have been defined to carry information about the media object in general: OCI descriptors and OCI streams. The following OCI descriptors are available [MPEG4-1, Clause 8.6.18]:

- ContentClassificationDescriptor
- KeywordDescriptor
- RatingDescriptor
- LanguageDescriptor
- ShortTextualDescriptor
- ExpandedTextualDescriptor

- ☞ ContentCreatorNameDescriptor
- ☞ ContentCreationDateDescriptor
- ☞ OCICreatorNameDescriptor
- ☞ OCICreationDateDescriptor
- ☞ SmpTeCameraPositionDescriptor
- ☞ SegmentDescriptor
- ☞ MediaTimeDescriptor

Content is classified, for example, by genre. Classification schemata need to be registered with a registration authority to make sure that all implementers of MPEG-4 systems have access to this information. The same approach holds for the rating descriptor that is supposed to indicate the suitability of the content for different groups of audiences. Descriptors for keywords, short text, and expanded text are all freestyle but presumably in increasing order of complexity. Name of content creators and OCI creators as well as the creation dates for those are self-explanatory. The camera position descriptor inherited from SMPTE [S315M] allows keeping track of the camera position for that media stream. Finally, the segment descriptor and media time descriptors are additions that allow assigning a name to a temporal segment of the media stream based on the definition of a media timeline (see section on synchronization).

Any of those descriptors can be attached to an OD. This means that all the ESs referenced through this OD share the same (static) object content information. The only information that can be attributed to a single stream within one single OD is the language, because it should be possible to convey different language variants of the same content as one logical media object, as presented in Figure 3.11. This is why the `LanguageDescriptor` features as a subdescriptor of the ESD (see Table 3.3).

OCI also may be dynamically changing over time—for example, the keywords and textual descriptor. In that case, a separate OCI stream may be used to convey that information [MPEG4-1, Clause 8.4]. Each OD can refer to one OCI stream at most. Within an OCI stream, an `OCI_Event` message is used to wrap each set of descriptors that is to be conveyed jointly. The `OCI_Event` has the added functionality that the embedded set of OCI descriptors can now be associated with a temporal segment of the media stream, identified through start time and duration (see Figure 3.12). The `SmpTeCameraPositionDescriptor` is a typical example of a descriptor that mostly makes sense when conveyed in an OCI stream, as it may be assumed that camera position changes frequently over time.

Concluding, it should be mentioned that OCI was established at a time when no other MPEG metadata format was available. Today the standardization effort for the MPEG-7 multimedia description framework is well advanced [N4509]. All concepts expressed in OCI could equally well be expressed in MPEG-7. In fact, a label for MPEG-7 streams has already been reserved. So, instead of an OCI stream, an MPEG-7 stream can be associated

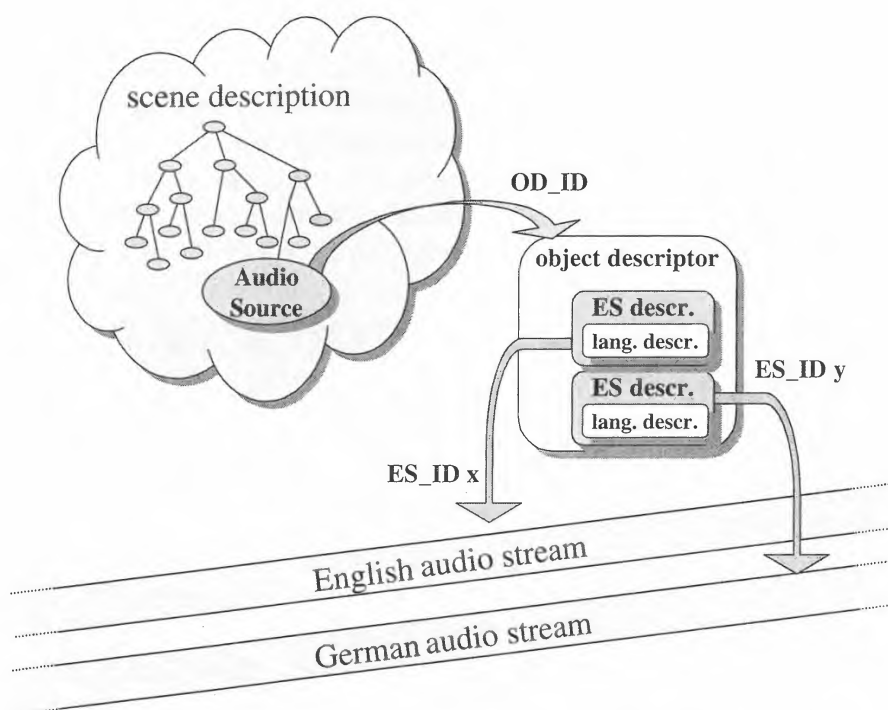


Fig. 3.11 Describing one audio object available in different languages.

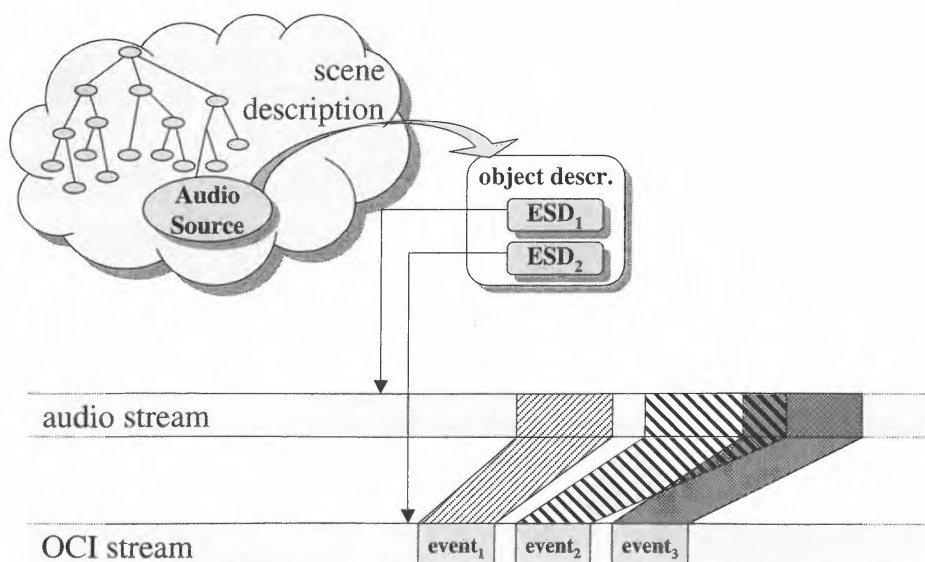


Fig. 3.12 OCI stream with various OCI events.

with an MPEG-4 media stream. It is hard to predict how OCI and MPEG-7 will coexist. It is conceivable that future MPEG-4 applications will rely more on MPEG-7 than on MPEG-4 OCI as a metadata format because of its larger capabilities, but it may also be that applications with few metadata requirements continue to use OCI.

mined by the maximum bit rate of the associated ES, as indicated in the ES descriptor. As with the basic SDM compliance, it is the sender's task to ensure that the FlexMux buffers never overflow. Or, phrased positively, the sender knows how to interleave data coming from different streams, being sure they will arrive in time, relative to all the other streams.

In Section 7.4.3.5, it will be seen how this model is further extended to cover the whole MPEG-4 over MPEG-2 delivery stack.

Both the FlexMuxBufferDescriptor and the FlexMuxTimingDescriptor need to be conveyed *out-of-band*—that is, outside the actual FlexMux stream, and, as for the MuxCodeTableEntries, this will be specified in the MPEG-4 adaptation of each DMIF instance. A proposal for in-band transport of these descriptors has been made, as that would give the additional benefit of establishing the FlexMux stream as a self-supported exchange format for multiplexed MPEG-4 streams, including delivery timing, independent of any delivery layer.

A potential use for in-band descriptor delivery would be applications that need to convey the same preproduced presentations both on MPEG-2 and on IP, or even those that need to forward content arriving on one delivery stack to another one. That would be largely eased, as a necessary gateway could simply act as a media pump, retrieving data from one delivery instance and pumping it out again through the second delivery instance, driven by the time stamps embedded in the FlexMux stream and, of course, taking into account the information in the embedded descriptors.

So, the rationale for in-band transport of FlexMux descriptors is the ease of streaming MPEG-4 presentations across heterogeneous networks. A generic solution to ease the exchange of MPEG-4 presentations, not necessarily in a streaming manner, already exists and is discussed in Section 7.3.

7.3 MPEG-4 FILE FORMAT

Historically, prior to the MPEG-4 standard, MPEG has not had an explicit file format. MPEG-1 and MPEG-2 content typically is exchanged as files that represent, in a sense, a stream ready to be delivered. These files usually have embedded absolute time stamps, and the data often has been fragmented with a particular transport in mind (e.g., MPEG-2 Systems transport stream). These characteristics can make random access difficult and editing or reuse of the streams hard without decoding, demultiplexing, and then rebuilding the stream after editing. In MPEG-4, no single transport protocol is *preferred*, so fragmenting the data to suit the packet size of a specific protocol was not acceptable.

The MPEG committee sought a *life-cycle* file format—one in which the files could be used when capturing media, editing it, and combining it; when serving the media as a file download or as a stream; and when exchanging partial or complete presentations. This need for a life-cycle format is not met in many simple file format designs. For example, as noted previously, the

design approach of MPEG-2, in which a stream is simply recorded to a file, makes editing hard. A more flexible design and approach were needed.

The design approach and principles embodied in the QuickTime file format from Apple [QUICKTIME] were closest to meeting these needs. After a requirements definition phase and a call for proposals, both design principles and details from QuickTime were adopted and refined to build the *MP4 file format*.

Perhaps the most unusual aspect of the MP4 design is the separation of the media data itself—the video data, for example—from data about that media data, often called *metadata*. Metadata might include timing information, the number of bytes occupied by a video frame, the location of the file in which it is stored, its position within that file, and so on. This timing and structural data is kept separate from the media data itself, in compact tables. So, for example, neither the length in bytes of a video frame nor its time stamps are stored adjacent to that frame, but in tables in the metadata part of the file. This is what is meant when the MP4 format is described as *nonframing*.

To further facilitate editing, the timing data is relative, not absolute. This means that time stamps do not have to be rewritten when, for example, a frame is inserted or deleted. Also, the interrelationships between streams are compactly represented in the metadata. In addition, media data is stored in its *natural* or base state, not preferring any one transport protocol or system: This means, for example, video frames are stored whole, not fragmented to any fixed size.

When the data must be streamed, special instructions for the protocol, optionally stored in the file, instruct streaming servers in the process of fragmenting and time stamping the data. These instructions are known as *hints*. As a result, the MP4 file format is, on first inspection, more complex than simple single-function file formats. However, as a format, MP4 is also more powerful than single-function formats and can be used for a wide variety of purposes.

With these principles in mind, this section looks first at how time is managed, then at how storage and space are managed in the context of the MP4 file format. Then the way MPEG-4 Systems concepts and structures are handled in the file format is explained. The special support for streaming is then detailed, and the MP4 section concludes with a printout of a small example file. This printout provides an example that can be reviewed as the format is described.

7.3.1 Temporal Structure: Tracks and Streams, Time and Durations

An MPEG-4 presentation is composed of a number of streams. In MP4, each stream is stored as a separate track in a file. A *track* in a file represents a timed series of media—successive video frames, for example. Each track has a *track identifier*, which maps to the `ES_ID`. A track is treated as a sequence of

samples, which are AUs, in MPEG-4 terms—the smallest entities to which timing can be attributed (see Chapter 3). So, a sample in a video track is a coded frame or VOP; in an audio track, a compressed frame of audio (not a single uncompressed audio sample); in a BIFS track, a set of instructions for one-time instant; and so on.

For each sample, the file format records the time *difference* between its time stamp and the time stamp of the next sample—its *duration*. This relative timing makes the process of editing much easier, as sections of the timeline may be moved around without disturbing their internal structure. Indeed, the file format has a facility for expressing precisely this operation. A set of structures known as an *edit list* can provide an explicit *map* of how the timeline of a single track should be mapped into the timeline of the presentation. Edit lists allow for the insertion of *empty time* and the use (and reuse) of sections of the timeline. So, a repeated piece of media can be defined once and used repeatedly by reference.

MPEG-4 video follows earlier MPEG standards and uses the concept of bidirectionally predicted frames or VOPs, called B-frames or B-VOPs. To decode a B-frame, the decoder needs three frames: the closest non-B-frame that will be displayed before it (called here $F[p]$), the closest non-B-frame that will be displayed after it (called here $F[s]$), and the B frame being decoded itself (called here $F(b)$). The decoder must be given these frames not in the order they are displayed but in the order that they must be decoded: Frame $F(s)$ is given to the decoder before the B-frame $F(b)$. To support this, and to manage decoding buffers in some terminals, MPEG separates the concepts of decoding time and composition (display) time. In the MP4 file format, frames are stored always in decoding order, and by default the decoding and composition time stamps are the same. However, an optional table can provide a positive offset from the decoding time to the composition time, when they differ. The reordering for display purposes is managed by this composition time offset.

Each track has a time scale, which defines how fast its clock ticks. Using an appropriate time scale enables precise definition of timing. For example, usually the time scale of an audio track will be chosen to match its sampling rate, to enable sample-accurate timing (e.g., a time scale of 22,050 ticks per second for 22.05 kHz sampled audio). In video, a time scale of 30,000 ticks per second, with media sample duration of 1,001 ticks, exactly defines NTSC video (often, but incorrectly, referred to as 29.97 frames per second).

In many sequences of media, some of the samples (frames) can be decoded without knowing any other samples: In video, these are known as I-frames or I-VOPs. In general, such samples provide useful random access or synchronization points. The indices of these samples are stored in a table, for each track, to enable random access and editing.

Sometimes, when seeking, it can be useful to find extra random access points. The file format provides a facility called *shadow sync* samples. These

samples provide an alternative random access point (I-frame or sync sample) for a sample that is not a random access point.

Streams stored in the same file are normally assumed to be synchronized together. However, the file format provides explicit facilities to indicate the synchronization relationships (clock reference relationships) between streams, if that is needed. This is done by means of *track references*, which are described in more detail below.

When a file is delivered as an MPEG-4 stream, the timing is conveyed by the MPEG4 SL, or by the transport layer being chosen (such as RTP on TCP/IP networks). Because MP4 files do not record the actual time stamps (as noted previously), the time stamps must be computed. This computation proceeds in three steps:

1. If the track has an edit list, this list is used to build an overall time map of the stream times from the implicit timeline of the sample times.
2. The decoding time stamp of each AU is then computed by adding the *durations* of the preceding samples in that track; this time is then mapped according to the edit list.
3. If the track has explicit composition time information, it is represented by the offset from the decoding time computed in step 2. The composition offset must be added to the decoding time, to obtain the composition time, and then again mapped through the edit list information (if any).

This process is shown in Figure 7.10. In Figure 7.10, a simple timeline of 10 samples is shown with an edit list. The edit list contains four edits: an empty edit, then two edits each containing three samples, and finally an edit containing two samples (marked empty, 3 sample edit, and 2 sample edit, respectively). The computation of the time stamps for the second sample involves (a) the duration of the empty edit; (b) the duration of the preceding sample in the edit, which gives the decoding time for this sample; and (c) the composition offset for this sample, which, added to the decoding time, gives the composition time.

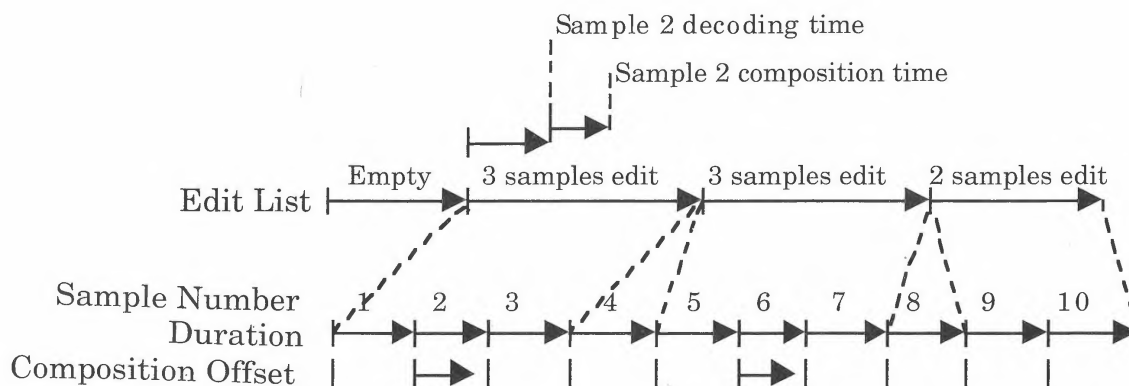


Fig. 7.10 Calculating time stamps for a sample.

7.3.2 Physical Structure: Atoms and Containers, Offsets and Pointers

This section introduces the concepts that underpin the physical structures in the MP4 file. All data stored in MP4 files is organized in atoms. An *atom* is a container that has an explicit *type* and *length*. So, for example, the synchronization points discussed previously can be found in a table stored in each track, in an atom of type `stss`; this atom contains a table listing, in ascending order, the numbers of the samples that are random access, or sync, points.

Atoms can contain other atoms. Typically, an atom either serves as a container or has fields that define particular concepts, although this is not a general rule. Any media data (samples) are stored in a media data atom (of type `mdat`).

All the metadata in a presentation is stored in the *movie* structure, the atom of type `moov`. Unused space in a file is designated with an atom of type `free`. The three atoms `mdat`, `moov`, and `free` are generally the only three atoms that occur at the top-level of a file. Of these, only the `moov` atom has internal structure. Inside the `mdat` atom, the media data itself is stored. Clearly the content of a `free` atom is irrelevant.

The `moov` atom contains a header atom and then a set of tracks. The movie header (`mvhd`) gives basic information about the presentation—the date it was last edited, for example, and its overall duration. The metadata for each track is stored in an atom of type `trak`.

Like the movie header, the `trak` atom has a track header, which also has creation and edit dates, a time scale for this track, and the overall duration of the track. This `trak` atom in turn contains a series of atoms, which define the media and how it may be handled.

There are four important structures in the `trak` atom. They are the edit list, the data reference information, the handler information, and the sample table.

The *edit list* is related to timing and was discussed previously. The *data reference information* defines whether the media data used in this track is stored in the same file or in other files (referenced by URL). This ability to refer to media data in many files is important when composing a project: Media from a library of files may be used by reference, and only copied when a self-contained file is needed. The *handler information* provides the first level of differentiation of the tracks; the handler type identifies the general kind of information in a track, such as visual, audio, BIFS, and so on. The *sample table* provides the detailed information about each sample—AU—within the track. It is structured as a set of atoms that are formatted as tables. These atoms define both the physical location of each sample (frame of video, for example) and its timing. To keep these tables compact, a variety of techniques is used. One that is used to compact the location and size information relies on the observation that several frames from the same track are often stored contiguously, even when data from various tracks is interleaved. This run of contiguous samples is called a *chunk*. The sample-to-chunk table provides the

mapping from sample number to chunk. The position of each chunk is recorded as a chunk offset (using 32 or 64 bits), which is measured from the beginning of the file in which the chunk resides. The length, in bytes, of each sample also is recorded, in the sample size table. Therefore, by using the data reference from the track, the chunk offset, and the sizes of the preceding samples in the same chunk, it is possible to find the data file containing the sample, which may be a file referenced by URL from the MP4 file itself, the chunk within that file, the offset of the sample within the chunk, and the size of the sample itself.

Figure 7.11 shows an example of a file that starts with the media data. In this media data, the frames of video and frames of audio are stored. There are three video chunks, containing three, two, and two samples (frames or VOPs) each. There are two audio chunks, containing two samples each. In this example, there also is some unused data. This unused data might be a video frame that has been edited and now stored elsewhere in the file, or any other kind of data that is no longer needed.

Note that because of the way this works, there is no need for the samples to use all the data in a file: There may be unreferenced data between the chunks. This can be used to skip unwanted data or the headers of a foreign file format, for example. Nor do the chunks have to occur in any specific order; data can be collected from disparate places in the files as it is needed. This means that much editing can be done by reference, simply changing tables and pointers.

The final structural concept that needs to be introduced is that of a *sample description*. A terminal wishing to display an MPEG-4 presentation needs to know how to decode each track: not only whether the data is visual or audio, but also how to set up each decoder to *understand* the coded data (e.g., whether the audio is AAC or HVXC, for example). This means knowing its corresponding audio or visual object type and decoder specific information. As with all the metadata encountered so far, this information is kept in a table. There is a set of sample descriptions in each track, and each sample is implicitly tied to exactly one of them. The name of the sample description *names* the

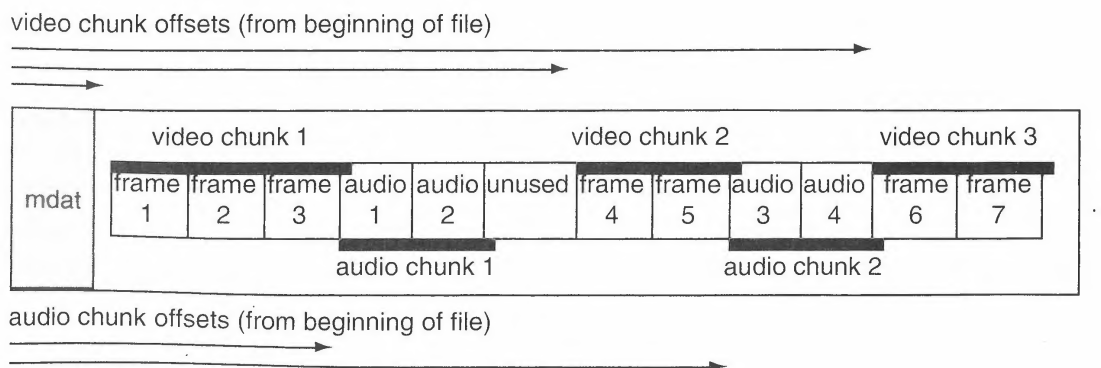


Fig. 7.11 Chunk offsets.

media format, and the parameters to that sample description parameterize the decoder as needed.

7.3.3 MPEG-4 Systems Concepts in MP4

So, how do the general concepts introduced previously map into the concepts introduced in MPEG-4 Systems? In order to facilitate editing, the descriptors for each stream are stored in the track that contains the stream. This makes it easy to move a stream from one file to another, for example. However, when a presentation is streamed, these descriptors are delivered to the terminal in an OD stream, or in the IOD (see Chapter 3). To avoid duplicating the descriptors, a special technique is used inside MP4 files: Neither the IOD nor the OD stream stored in the file contains ES descriptors (ESDs). Instead, a reference to the appropriate tracks is placed in the ODs.

A new MP4 structural concept is used here. Each track has a table of typed references to other tracks. The OD track has track references to those tracks for which it will carry setup information. In that OD track, at the time when an ESD is needed, a structure is stored instead which refers to the track from which the ESD is needed. In the process of streaming, the reference is replaced by the actual descriptor from the referenced track.

The initial *hook* into an MPEG-4 streamed presentation is the IOD. This descriptor, too, would normally contain the ESDs for the streams that must be initially decoded. As with the OD stream, these are not stored in the IOD; instead, again, the track ID of the initial tracks is stored in the file, and these structures must be replaced by the appropriate ESDs when the file is streamed.

The other MPEG-4 streams (and, indeed, streams that are integrated with MPEG-4, such as MPEG-7) follow these structures without needing any specific support from the file format. For example, BIFS streams—both animation and update—use these structures, including the concept of random access, or sync, samples. A random access point in BIFS performs a *Replace Scene* command.

7.3.4 MPEG-4 Track Types and Storage

In the MP4 file format, determining how to decode a track involves two levels of decision. First, what basic kind of track is it: audio, visual, and so on? And then, precisely how are the samples coded: what codec—MPEG-4 object type—and with what parameters and settings (decoder-specific configuration)?

The first decision is made by examining the *handler type*; along with the handler type, there is a type-specific header atom within the track structures for each kind of track. The code points shown in Table 7.5 are used for the handler type and the type-specific header atom.

The last line in Table 7.5 is included for completeness; clock reference streams are not usually stored within the file. Instead, if a particular stream-

Table 7.5 MP4 handler and header atom types

Stream type	Handler type	Header atom type
VisualStream	vide	vmhd
AudioStream	soun	smhd
ObjectDescriptorStream	odsm	nmhd
SceneDescriptionStream	sdsm	nmhd
MPEG7Stream	m7sm	nmhd
ObjectContentInfoStream	ocsm	nmhd
IPMP Stream	ipsm	nmhd
MPEG-J Stream	mjsm	nmhd
Hint Track	hint	hmhd
ClockReferenceStream	crsm	nmhd

ing transport requires a clock reference stream, it is constructed when a file is prepared for streaming, or by the streaming server itself.

7.3.5 Hinting

The penultimate line in Table 7.5 declares hint tracks. These are not MPEG-4 media tracks at all; instead, they are the key to understanding how efficient streaming may be done from a file in which the media data has been stored without regard to the needs of a particular streaming protocol.

Streaming media data involves sending media to a client system *just in time*, using small amounts of buffering. If packets get lost in transit, the client usually has to carry on; even if a retransmission is requested, there is no assurance it will arrive in time. The way in which the decoder can cope with this loss is critically dependent on the way the media data was originally placed into packets. For example, if a media sample (such as a video frame) is split arbitrarily into packets, any one of those packets being lost will usually make the rest of the frame, or perhaps the entire frame, unusable, and packets that did arrive will have to be discarded.

However, this problem may be alleviated by packetizing the media data carefully. If appropriate *boundary* points are found in the media, then the client often can recover all the data from the packets that successfully arrived. This can make a major difference to the user experience. For example, if a stream divides each video frame into six packets, and there is 10% loss, almost half the frames will lose at least one packet ($1 - 0.9^6$). If all those frames are discarded, the user experience is of very degraded video (approximately 50%

missing). If, instead, the parts of those frames that arrive are displayed, the user experience is of the full frame rate, with 10% degradation.

The problem in finding these boundary points is twofold: Scanning the media data to find these boundaries is time consuming, and the way to do it requires the knowledge of the coding system being used—format parsing. If a media server must execute these rules as it streams, it will slow down; and if new rules are introduced, then every server must be updated.

Hint tracks provide a bridge between the media data and the packet data. In a media track, a sample is an audio frame—a media sample. In contrast, a hint sample tells the server how to make a packet or set of packets. Hint samples reference the media tracks they are packetizing and contain instructions, such as “insert these 4 header bytes into the packet,” “copy 200 bytes from offset 43 in sample 267 of my media track,” and so on. By following these instructions, the server can rapidly build valid packets without knowing either the coding system or the packetization rules for that coding system. Special authoring tools called *hinters* generate these hint tracks, after the editing process is complete. Hinters are specific both to the format of the media data and to the requirements of a specific transport protocol. The format-specific knowledge required has been moved from the server to the hinter. This means that the hinter is at liberty to compute the boundaries carefully and perform other optimizations, transformations, or corrections that are needed. So it would normally be the hinter that constructs any required clock reference stream, for example. And likewise the hinter would transform the OD track into an OD stream, replacing track references by the required ESDs from those tracks.

Of course, servers always have the freedom to know the parsing rules and packetize on the fly; this is why they are called hints—hints that the server is free to ignore.

Hint tracks are structured like media tracks. Just as media tracks must cope with differing coding systems, so hint tracks must manage multiple streaming protocols. Each server must find the hint tracks for the protocol it uses, and ignore the rest.

As a result, the same file contains both the original media—unpacketized, in its native format—and support for potentially multiple streaming formats. Indeed, by careful use of the *data references*, various MP4 files can be built that use the same media database in different ways. A hinted file can always be taken from a streaming server, the hint tracks deleted, and then edited, rehintered, and replaced on the server. This is part of the life-cycle management that MP4 offers.

The format of the hint samples is, of course, protocol-specific. Because this parallels the way that the format of media samples is codec-specific, the same structure is used. The sample description both names the protocol (and hence defines the format of the hints) and provides any necessary parameters to the server.

At the time of writing, the format of hint tracks for the RTP protocol is being standardized. There is also a proposed format of hint tracks for MPEG-2 Systems transport streams.

7.3.6 Atoms

At this point, all the basic MP4 concepts are in place: how time and space are handled, how MPEG-4 structures are mapped into the MP4 file format, and the basic concept of hinting. This section presents the atoms that make up an MP4 file.

All atoms start with a 4-byte `size`, and a 4-byte `atom type`. The type is usually four printable characters (this makes debugging and inspection of files much easier). The `size` is the length of the entire atom in bytes, including this 8-byte header. Two special values of the `size` field are relevant: 0 means that this atom extends to the end of the file, and 1 means that a 64-bit `size` field immediately follows the `type` field. This allows for an occasional very large atom.

Many atoms are defined to have a `version` and `flags` field, of 1 and 3 bytes respectively, following the `type`. These are called *full atoms* in the specification. A few atoms have more than one version, and use the `version` field to indicate which variant is present. The `flags` field is used by some atoms for purposes specific to that atom.

7.3.6.1 Top-Level Atoms Three kinds of atoms occur at the top-level of MP4 files (not contained in any other atom):

1. **moov:** The movie atom `moov` is the container for all the metadata in the file. A movie atom is mandatory in an MP4 file; it can be anywhere in the top-level sequence of atoms. The `moov` atom is a pure container atom; it has no fields of its own, and it contains other atoms.
2. **mdat:** If there is media data in this file, it is stored in an `mdat` (media data) atom.
3. **free or skip:** If there is spare space, that is in atoms of type `free` or `skip`.

7.3.6.2 Inside the Movie Atom A movie atom `moov` contains these atoms:

- ☞ A movie header atom `mvhd`
- ☞ Optionally, an IOD atom `iods`
- ☞ A track atom `trak` for each track in the presentation

The movie header has bookkeeping fields for the creation and modification time of the movie itself. It also records the duration of the presentation (the duration of its longest track, considering edits), and defines the time scale (in ticks per second) for the movie and track duration values. Finally, it has a helper field that suggests the next free track identifier. The track atom is a

Natural Video Coding

by Michael Wollborn, Iole Moccagatta, and
Ulrich Benzler

Keywords: object-based video coding, motion compensated hybrid video coding, shape coding, error resilience, scalability, texture coding

The coding of natural video content, including still images, has been under investigation in research and industry for a long time. But what does it mean, to *code* video content? In early times, the only goal was to represent a video scene, be it moving or not, in a format allowing transmission or storage. And, because storage capacity and bandwidth were and still are restricted resources, the main and ultimate objective of video coding for a long time was *compression*. In this sense, the coding of video content aimed mainly at representing it using as few data as possible, at a certain visual quality, or vice versa, to maximize the visual quality at a given bit rate.

This coding paradigm is reflected in the early video coding standards, such as H.261 [H261] and MPEG-1 Video [MPEG1-2]. Of course, they also included application-driven functionalities such as random access, scalability, and others; however, the idea of improving compression was always the driving force behind all the activities. At the same time, several research activities [e.g., Kunt85, MuHO89, SaTM95] investigated and proposed a different approach than the well-established *coding of rectangular frames*—namely, the so-called *object-based* or *region-based* video coding. Although one objective of these proposals was higher compression, the principal ideas opened a much broader scope of possible functionalities and applications. Influenced by both

the well-established video-compression standards and the new ideas of coding (not the rectangular window to the visual world, but its content), the MPEG-4 natural video coding activities began in 1995 with the aim of combining the benefits of both approaches: high compression over a broad range of bit rates and innovative content-based functionalities.

This chapter gives an overview of the MPEG-4 natural video coding tools [MPEG4-2]. However, it does not describe the complete standard in all details—that would require a book of its own. Therefore, the emphasis is on the general concepts and on those tools that are newer, compared to already existing technology and standards. For more details on specific topics, the reader is referred to the references.

8.1 GENERAL OVERVIEW

In this section, an overview of the MPEG-4 natural video coding tools is given. The functionalities and respective application scenarios are described briefly, followed by a short description of the principal MPEG-4 video coding approach.

8.1.1 Functionalities and Application Scenarios

The MPEG-4 Visual standard [MPEG4-2] comprises tools for coding natural visual sources—that is, sequences of images (video)—as well as still images (visual texture). Whereas the video coding is based on the well-known motion-compensated hybrid DCT coding scheme, the visual texture coding (VTC) is based on wavelet transform and zero-tree coding.

In addition to previous standards that addressed only coding efficiency, the MPEG-4 video coding tools provide several additional functionalities (see also Chapter 1): coding of arbitrarily shaped objects; efficient compression of video sequences and still images over a wide range of bit rates; spatial, temporal, and quality scalability; and robust transmission in error-prone environments.

These advanced features greatly aid content creators in generating rich video and thus multimedia content. In particular, the ability to *code objects of arbitrary shape and size* goes beyond the scope of all previous video and image coding standards. It allows content creators to overcome the limits of frame-based content coding and opens the doors for a multitude of innovative applications in the future.

The ability to efficiently compress still textures and videos over a wide range of bit rates using the MPEG-4 tool set greatly simplifies the content creator's job by providing one-stop shopping opportunity for nearly lossless to lossy compression while delivering the best quality for the bit rate required by the application.

Scalable video coding is of interest for various applications. For mobile communication, a hierarchical transmission system can be used, in which the

base layer signal is especially well protected against errors in the transmission channel by applying robust forward-error correction to the base layer bitstream. For video server applications, scalable coding allows users to access the video at different bit rates, without the need for multiple coding and storage of separate video bitstreams in different resolutions or qualities.

To enable the transmission over wireless and wired networks with severe error conditions, the MPEG-4 Visual standard has adopted *error-resilience coding* tools. Depending on the transmission channel, these tools can replace or complement channel coding to improve quality of service (QoS) [M4575].

Finally, the MPEG-4 VTC provides an efficient tool for the compression of still images, based on the wavelet transform and zero-tree coding. It can be combined with the MPEG-4 shape coding tools and thus provides, with this respect, the same features as the video coding tools. The combination of VTC with the natural video coding tools allows the coding of scenes comprising both moving and still images by means of a single standard.

8.1.2 Basic Principles

One important principle in ISO/MPEG standardization activities is that the encoder is not standardized—only the bitstream it produces and the procedure for decoding it (i.e., the decoder). The reason is that for interoperability, it is not necessary to standardize the encoder. As a positive consequence, there are several degrees of freedom and optimization left for building the encoder. This gives one the possibility of improving the performance of the coding algorithm by introducing new (non-normative) techniques while the standard is already fixed, and it also leaves room for competition in building better encoders. This principle holds true also for MPEG-4 natural video coding. Therefore, in the remainder of this chapter, mostly decoding procedures will be described. Only when it is essential for the understanding of the technical issues will encoding algorithms be explained.

As mentioned in the introduction, MPEG-4 natural video coding tools comprise a whole tool-set for different applications and use cases. Moreover, these tools can be combined to code different sources of natural video. Not all of the tools are required for every application, and in some cases, it may be preferable if a decoder not support certain tools (e.g., for reasons of complexity or computational load). On the other hand, for reasons of interoperability, a decoder should at least support a clearly defined set of tools. Thus, for the MPEG standards, *profiles* are defined. For each profile, the tools to be supported are specified, and each decoder that claims to be compliant to a certain profile must support all of these tools. Further, for each profile different *levels* are specified. A level sets certain complexity bounds—for example, for the required memory, number of objects, or the bit rate that can be handled by respective decoders. In order to define a conformance point for an MPEG decoder, in addition to the profile the level must be specified. This means that

for compliant decoders a profile and a level must be defined. More details about MPEG-4 conformance and profiling can be found in Chapter 13.

In Figure 8.1, the complete set of decoders described in the MPEG-4 Visual standard [MPEG4-2] is shown. The decoders printed in gray (face and body animation decoding and [2D and 3D] mesh decoding) are not described in this chapter, but in Chapter 9. Also the scene composition is not explained here, but in Chapter 4. All of the other elements shown in Figure 8.1 (except for the DeMUX, which is a simple demultiplexer) are explained in detail in the following sections.

Analyzing Figure 8.1, several MPEG-4 natural video coding features can be highlighted. First, it is possible to code both still images (referred to as visual textures) and video sequences with the standard's tool-set. Moreover, taking only the two blocks Motion Decoding and Compensation and Texture Decoding results in the structure of the well-known (frame-based) motion-compensated hybrid coder—actually, it is even a block-based one. Adding Shape Decoding to the tool-set converts the entire MPEG-4 Visual standard into an object-based video coding solution. This means that the coded content is no longer restricted to be rectangular, but can be of arbitrary shape, which is coded by means of the MPEG-4 shape coding tools. In this mode, the shape is used also for decoding of motion and texture and for motion compensation.

It may seem that the usage of shape, and thus the possibility for content-based functionalities, is the only innovation of MPEG-4. However, this is not the case; the *rectangular* or *frame-based* parts of the video coding algorithms also have been greatly improved—for example, with increased compression

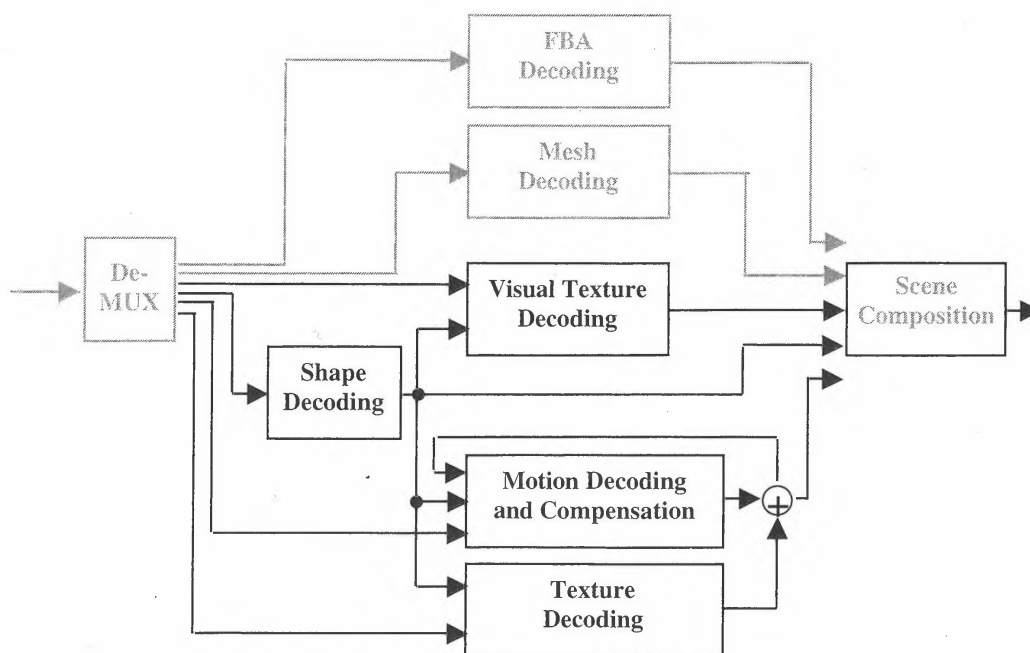


Fig. 8.1 Overview of MPEG-4 natural video decoding tools.

efficiency and error resilience. This will be described in detail in the remaining sections. Nevertheless, the object-based approach is a major issue in MPEG-4, and therefore the basic ideas and principles are described in the following section.

First, it has to be noted that even in the object-based mode, the coding of the images of a video sequence in MPEG-4 is done blockwise. This means that each image is subdivided into square blocks, whose content is then coded. In order to understand this principle, some terminology for MPEG-4 video coding is introduced and explained, using the simple exemplary scene shown in Figure 8.2.

In MPEG-4, each scene to be coded can be composed of one or several *visual objects* (VOs). In case of video sequences, the VO is of type *video* and is defined as a sequence of *video object planes* (VOPs). In the following, a VO of type *video* is also referred to as *video object*. Each VOP is an instance of the video object at a certain time instance, t . This means that a video object is composed of a sequence of VOPs, much as an image sequence is composed of a sequence of images. Using that analogy, the VOP corresponds to a single image, and the video object corresponds to an image sequence. The main difference is that a VOP can be either rectangular or of arbitrary shape. In the latter case, it is defined not only by its Y, U, and V components but also by its shape.¹ The representation and coding of the shape will be described in detail later.

The complete hierarchy of an MPEG-4 video bitstream is shown in Figure 8.3. The top level is a visual sequence (VS) with several VOs. Each VO of type *video* comprises one or (in case of scalable video coding) more *video object layers* (VOLs). Each layer corresponds to a certain spatial resolution or image quality, as will be explained in Section 8.4. Further, a set of successive VOPs

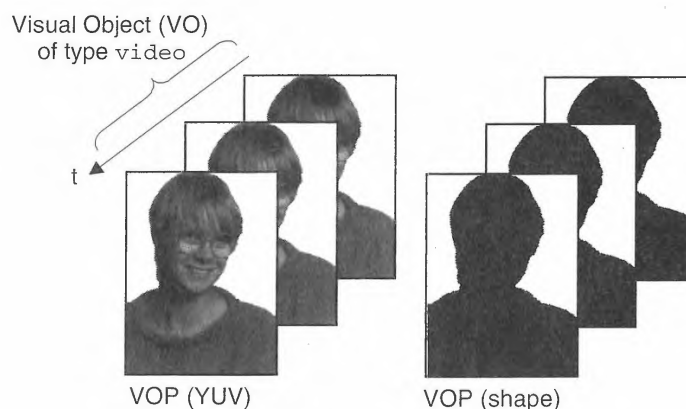


Fig. 8.2 Visual object (VO) of type *video*, represented by a sequence of video object planes (VOPs), which are composed of Y, U, and V texture matrixes and a shape matrix in case of an arbitrarily shaped VO, as shown. Only the nontransparent pels, marked black in the shape matrix, belong to the VO.

1. In other contexts, the VO shape matrix is also referred as the *alpha plane*.

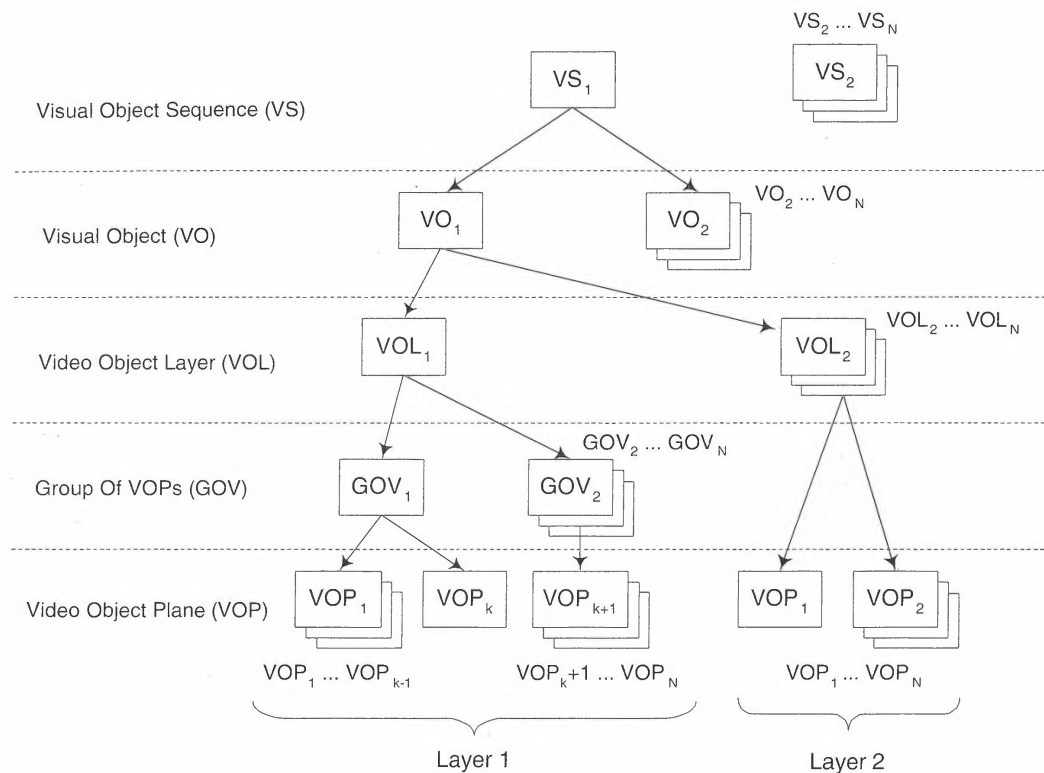


Fig. 8.3 Hierarchical structure of the MPEG-4 video bitstream.

can be clustered in order to form a *group of VOPs* (GOV). The GOV carries header information, which is useful for random access and resynchronization (e.g., in case of channel errors). The lowest level of the hierarchy is the VOP, as shown in Figure 8.3.

Just as previous coding standards do not specify how a video sequence is captured and even encoded, MPEG-4 does not standardize how a video object (and in particular the arbitrary shape information) is generated. The applied segmentation algorithm may be fully automatic (e.g., as described in [SalP94, NeCR97, MecW98]) or existing blue screen technology may be used. Also, user-assisted systems (as described in [AOWM98, MCMM99, KJKL99]) can be applied to generate arbitrarily shaped video content of high visual quality.

For coding purposes, each VOP is processed blockwise. Therefore, the smallest surrounding rectangle of the VOP is built, the *VOP bounding box*. This bounding box is extracted from the scene for each time instance and used to represent the corresponding VOP by means of its texture and shape information (see Figure 8.4).

For coding, the bounding box is divided into square blocks of size 16×16 luminance pels (and corresponding chrominances), the *macroblocks* (MBs). Therefore, the VOP bounding box is extended to match horizontal and vertical

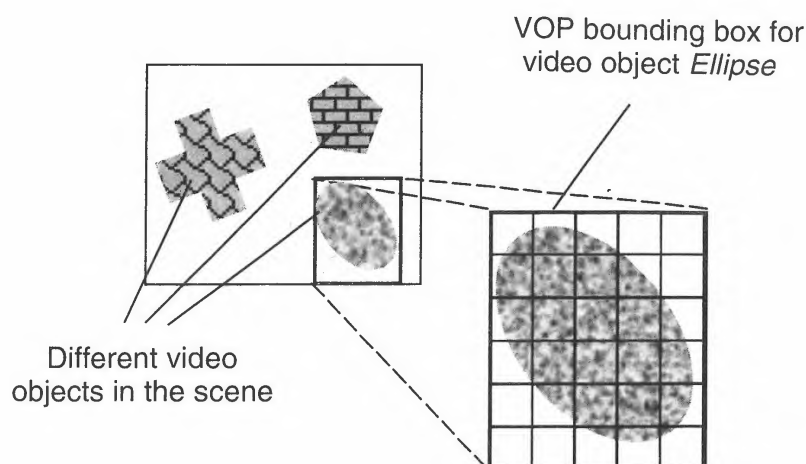


Fig. 8.4 Exemplary scene with three different video objects (*Cross*, *Pentagon*, and *Ellipse*); enlarged is the VOP bounding box for the video object *Ellipse*, divided into macroblocks of size 16×16 pels.

size of multiples of 16 pels (as the smallest surrounding rectangle to the object does not necessarily have multiples of 16 pels dimensions). Three different types of MBs are distinguished in the VOP bounding box, as shown in Figure 8.5.

1. **Transparent:** MBs that are completely outside the VOP. For these MBs, there is no YUV data to be coded; due to the shape information, the receiver knows that these MBs are *transparent* (i.e., not visible in the decoded scene).
2. **Opaque:** MBs that are completely inside the VOP. These MBs are processed as in the well-known block-based hybrid coders—that is, they are either *intra* coded using their YUV values or *inter* coded applying motion-compensated prediction and coding of the prediction error. The processing of these blocks is described in Section 8.2.

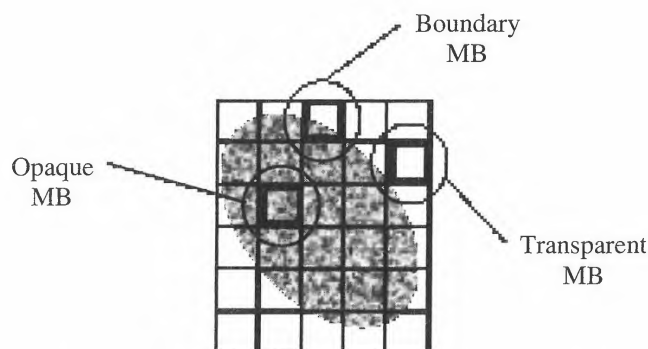


Fig. 8.5 MB types within the VOP boundary box.

3. **Boundary:** MBs that are at the boundary of the VOP. For processing these boundary MBs, specific tools for coding of arbitrarily shaped objects have been developed. These tools are presented in Section 8.3.

In addition to the two sections mentioned previously, which describe the MPEG-4 coding of rectangular video objects and of arbitrarily shaped video objects, there are sections describing specific MPEG-4 video coding tools with more detail. In Section 8.4, scalable video coding is explained. In particular, those parts of the scalable coding that are new compared to the state-of-the-art scalable video coding known, for example, from MPEG-2 Video [MPEG2-2], are emphasized. In Section 8.5, specific tools for interlaced video coding, error-resilient coding, dynamic resolution coding, sprite coding, and high-quality texture coding are described. Finally, Section 8.6 describes the MPEG-4 visual texture coding tool, which is an efficient wavelet-based tool for coding of rectangular as well as arbitrarily shaped still images.

8.2 CODING OF RECTANGULAR VIDEO OBJECTS

In MPEG-4 video coding, two main coding modes are distinguished: the coding of rectangular video objects and the coding of arbitrarily shaped video objects. In this section, the coding of rectangular video objects is described.

In principle, a rectangular video object is identical to a *frame* that is used in well-known video coding standards such as MPEG-1 and MPEG-2 [MPEG1-2, MPEG2-2, HaPN96], H.261 and H.263 [H261, H263, Rijk96]. Also, the coding principle of rectangular MPEG-4 video coding is the same as in these standards—namely, motion-compensated block-based hybrid coding. However, some significant improvements have been introduced by applying new tools or improving known algorithms. Therefore, this section describes the root coding algorithm only briefly, and concentrates on the new tools and algorithms introduced by MPEG-4 Visual [MPEG4-2].

8.2.1 Overview

For coding of rectangular video objects, only motion and texture information is transmitted to the decoder. Shape information is not required, and therefore in this case, no shape coding and decoding are necessary. In this respect, the MPEG-4 video coding solution is a block-based motion-compensated hybrid coder, as shown in the block diagram of Figure 8.6.

Figure 8.6 shows a block diagram of the decoding process for MPEG-4 video. Because no shape information is required for rectangular video objects, the corresponding modules in the block diagram are depicted in gray. The remaining blocks are those for motion decoding and compensation and for texture decoding. Compared with known block-based hybrid coding solutions, all

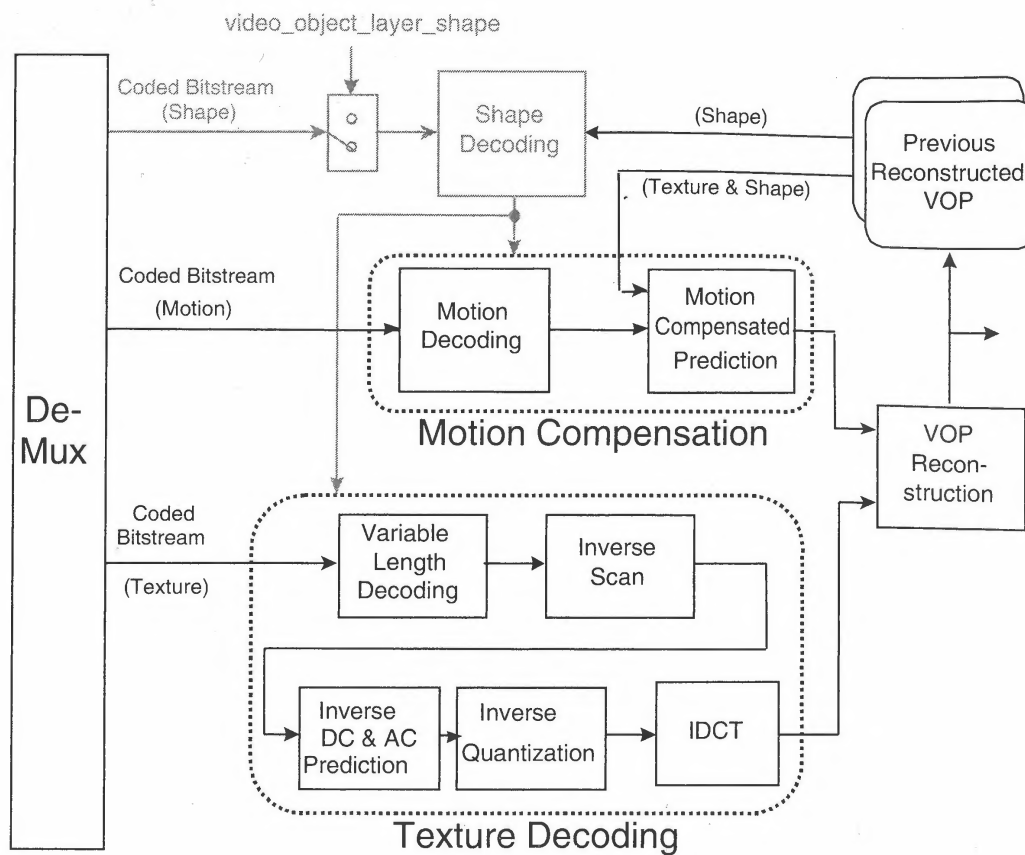


Fig. 8.6 MPEG-4 video decoding block diagram (shape decoding is in gray, as it is not required for rectangular video objects).

of these blocks have been significantly improved by introducing new or enhanced tools and algorithms.

Because the block-based hybrid coding principle is well known and copiously described in the literature (see, e.g., [CFHM71, Giro87, MuPG85, NetH88]), only a short description of the overall coding technique is given here. Each image of a video sequence is considered as one (rectangular) VOP. The coding of each VOP is performed blockwise—that is, each VOP is divided into macroblocks of 16×16 luminance pels size (with the corresponding chrominances, which depend on the colour subsampling format). Two major coding modes are distinguished:

1. **Intra mode:** The luminance (Y) and chrominance (U,V) values for each MB are coded independently of previous or future VOPs.
2. **Inter mode:** The difference of the MB luminance and chrominance values with respect to prediction values obtained from previous or future VOPs is coded.

The advantage of the inter mode is that, in general, the images in a video sequence do not change their content completely between two images (or VOPs, speaking in MPEG-4 terminology). Therefore, not the complete image but only the differences to the previous or future images (or combinations of them), referred to as the *reference image*, need to be transmitted to the decoder. Because the decoder knows the reference image(s), it can reconstruct the current image by adding the received difference signal. This principle is also referred to as *predictive coding*—that is, the encoder first calculates prediction values and then transmits only the *prediction error*.

In motion-compensated block-based hybrid coding, this prediction takes into account the motion within the captured scene and is therefore called *motion-compensated prediction*. The motion information is calculated and coded as side information, so that the decoder can carry out the motion-compensated prediction. The representation of the motion information depends on the specific coding algorithm. In MPEG-4, two motion modes are defined at the MB level:

1. **1MV mode:** One motion vector (MV) is transmitted for the complete macroblock (i.e., for each 16×16 luminance pels block).
2. **4MV mode:** The macroblock is subdivided into four blocks of 8×8 luminance pels, and one MV is transmitted for each of these blocks (i.e., in total, four MVs for the complete MB).

A VOP can be coded using one of two prediction modes: For *predicted VOPs* (P-VOPs), the reference image is the previously decoded intra coded VOP (I-VOP) or P-VOP, whereas for *bidirectionally predicted VOPs* (B-VOPs) both the previous and the subsequent I- or P-VOP can be used for prediction.

The computation of the motion information is referred to as *motion estimation*. Because it is an encoder issue, the standard does not specify how to carry out this step, as however it is done, it does not impact interoperability (providing that the syntax and semantics are respected); however, exemplary algorithms can be found in an informative annex of MPEG-4 Visual [MPEG4-2] and in the informative part of the MPEG-4 Reference Software [MPEG4-5]. Some specifically optimized algorithms can also be found in the MPEG-4 Optimised Reference Software [N4554].

For the coding of the texture (i.e., the luminance and chrominance information), the same algorithms are used for the intra and inter modes. The difference is that in the intra mode actual texture values are coded, whereas in the inter mode the coded values represent the prediction error (i.e., difference values). The texture coding is based on a 2D 8×8 *discrete cosine transform* (DCT; see, e.g., [NetH88, MPEG4-2]), with subsequent quantization and entropy coding of the transform coefficients. In addition, predictive coding is used for some of the transform coefficients, referred to as AC/DC prediction.

In the following sections, the motion-compensation tools and algorithms introduced by the MPEG-4 video standard are described. Then, the improvements regarding the texture coding are explained in detail.

8.2.2 New Motion-Compensation Tools

In this section, new and improved tools and algorithms for motion compensation in block-based hybrid coding, introduced by the MPEG-4 Visual standard [MPEG4-2], are described. Therefore, first the temporal prediction structure is described, as it is essential for the understanding of the following sections. Then, the following tools and algorithms are explained:

- ☛ **Quarter-pel motion compensation:** Algorithm for motion compensation using MVs with an increased resolution of one-quarter pel instead of only half- or full-pel resolution used in previous standards such as H.261, H.263, MPEG-1 Video, and MPEG-2 Video. This allows a significantly improved prediction and thus decreases the prediction error.
- ☛ **Global motion compensation:** A single set of motion parameters is coded for the complete VOP, representing the VOP global motion. These parameters can be used alternatively to the (local) MVs of an MB for motion compensation. This tool is particularly important for sequences with a large portion of global motion (e.g., caused by a moving camera) and for nontranslational motion like zoom or rotation.
- ☛ **Direct mode in bidirectional prediction:** This is an improvement of the bidirectional motion-compensated prediction, which uses the MVs of neighboring P-VOPs in order to decrease the bit rate required for coding the B-VOP MVs. It is a generalization of the “PB frames” introduced by H.263 [H263].

Those readers who are more familiar with the MPEG-4 Visual standard may miss one tool in this list—namely, the *overlapped block motion compensation* (OBMC). Because at the time of writing OBMC is not included in any of the MPEG-4 Visual object types and thus cannot be used by any MPEG-4-conformant application up to now, this tool is not described in this book. The interested reader is instead referred to the literature [MPEG4-2, OrcS94].

8.2.2.1 Temporal Prediction Structure In order to understand the algorithms and terminology of motion-compensated prediction, it is important to clarify the temporal prediction structure. For the MPEG-4 video case, this structure is depicted in Figure 8.7.

As shown in Figure 8.7, an I-VOP does not refer to any other VOP (has no reference VOP), as it is completely encoded using the intra coding mode. P-VOPs refer to the so-called *forward reference*, which is defined as the most recently decoded I- or P-VOP in the past for which information has been trans-

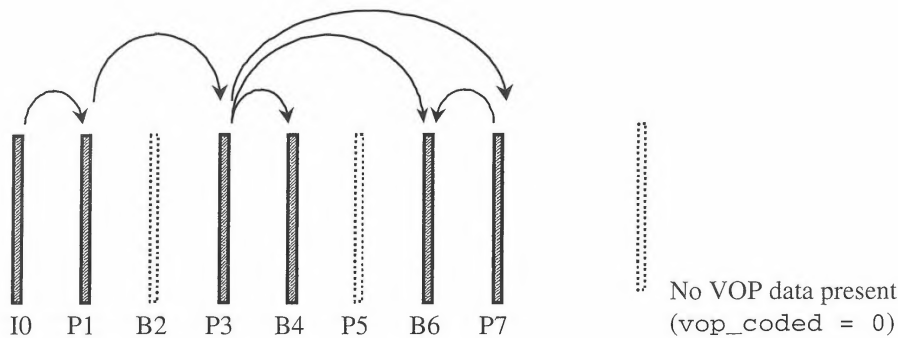


Fig. 8.7 Temporal prediction structure of MPEG-4 motion-compensated hybrid coding. Intra VOPs are marked as I_x , predictive VOPs as P_x , and bidirectionally predicted VOPs as B_x . The arrows describe the direction of the prediction. The number x denotes the display order of the VOPs, not their order of transmission [MPEG4-2].

mitted (signaled by the condition² $vop_coded=1$). Finally, B-VOPs may refer to the forward reference (as just defined) as well as to the *backward reference*, which is defined as the most recently decoded I- or P-VOP in the future. In the case that no information has been transmitted that can be used as backward reference (i.e., $vop_coded=0$ for the backward reference), only the forward reference is used for the prediction of the B-VOP.

In Figure 8.7, the *display order* of the VOPs is shown. It should be noted that this order is not identical to the *transmission order*. As the B-VOPs may make reference to an I- or P-VOP that is to be displayed *in the future* with respect to their own display time, these backward-reference VOPs must be transmitted before the corresponding B-VOPs. For the example shown in Figure 8.7, the transmission order would be I0 P1 P3 B2 P5 B4 P7 B6. Because in that example no information is transmitted for VOPs B2 and P5 (i.e., for these VOPs $vop_coded=0$), VOPs B6 and P7 make reference to VOP P3. The omission of VOP B2 has no influence on the referencing structure, as B-VOPs are not used as reference for motion-compensated prediction.

8.2.2.2 Quarter-Pel Motion Compensation The MVs that are used for the motion-compensated prediction must be transmitted to the decoder. Therefore, the resolution of the MVs must be limited to a finite number. In the early motion-compensation algorithms (e.g., H.261 [H261]), only full-pel MVs were used—that is, the components of an MV were restricted to an integer number. Because the real motion between two successive images of a sequence is, in general, not exactly described by steps of one pel, this restricted resolution leads to errors in the motion-compensated prediction and thus to an increased prediction error.

2. The parameter vop_coded is a syntax element that is transmitted at VOP level; if it is set to zero, this signals that no more information is transmitted for this VOP—that is, the VOP is “empty” (no VOP data present).

Edited by Fernando Pereira and Touradj Ebrahimi

THE MPEG-4

TK
6680.5
.P47
2002

THE MPEG-4 BOOK

Edited by Fernando Pereira and Touradj Ebrahimi

The most complete, focused guide to MPEG-4—the breakthrough standard for interactive multimedia

MPEG-4 represents a breakthrough in multimedia, delivering not just outstanding compression but also a fully interactive user experience. In **The MPEG-4 Book**, two leaders of the MPEG-4 standards community offer a comprehensive, targeted guide to the MPEG-4 standard and its use in cutting-edge applications. Fernando Pereira and Touradj Ebrahimi, together with a unique collection of key MPEG experts, demonstrate how MPEG-4 addresses tomorrow's multimedia applications more successfully than any previous standard. They review every element of the standard to offer you a book that covers:

- Synthetic and natural audio and video object coding, description, and synchronization
- BIFS—the MPEG-4 language for scene description and interaction
- The extensible MPEG-4 textual format XMT
- Transport and delivery of MPEG-4 content
- MPEG-J: using Java™ classes within MPEG-4 content
- A complete overview of MPEG-4 profiles and levels
- Verification tests

The authors also walk through the MPEG-4 Systems Reference Software—offering powerful real-world insights for every product developer, software professional, engineer, and researcher involved with MPEG-4 and state-of-the-art multimedia delivery.

The MPEG-4 Book is part of the new IMSC Press Multimedia Series from the Integrated Media Systems Center at the University of Southern California, a federally funded center specializing in cutting-edge multimedia research.

About the Editors

FERNANDO PEREIRA, Associate Professor at Instituto Superior Técnico, Lisboa, Portugal, and researcher at Instituto de Telecomunicações, is area editor of *Signal Processing: Image Communication Journal* and Associate Editor of *IEEE Transactions of Circuits and Systems for Video Technology*. Pereira has chaired many ad hoc groups in the context of the MPEG-4 and MPEG-7 standards' development and received an ISO certificate for his contributions to the MPEG-4 standard. His current interests include video analysis, processing, coding, description, and interactive multimedia services for Internet and mobile environments.

TOURADJ EBRAHIMI, Professor of Image Processing at EPFL's Signal Processing Lab in Switzerland, has been nominated for two ISO certificates for his contributions to MPEG-4 and JPEG 2000 standards. He leads the Swiss delegation to MPEG and JPEG and is associate editor of *IEEE Transactions on Image Processing*, area editor of *Image Communication Journal of EURASIP*, and associate editor of *Optical Engineering*. His interests include image processing, coding, and visual information security. He holds a dozen patents.

IMSC Press Multimedia Series/Andrew Tescher, Series Editor



PRENTICE HALL

ISBN 0-13-061621-4



9 0000

