Design of the Integrator to Work with HyTime

by

Arun Narasimhan

Thesis submitted to the faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

in

Computer Science and Applications

©Arun Narasimhan and VPI & SU 1992

APPROVED:

Dr. Edward A. Fox, Chairman

Dr. Deborah S. Hix

Mr. Edward E. Schwartz

January, 1992
Blacksburg, Virginia
Design of the Integrator to Work with HyTime

by

Arun Narasimhan

Committee Chairman: Dr. Edward A. Fox
Department of Computer Science

(ABSTRACT)

In order to produce an Interactive MultiMedia Application (IMMA), the developer of that application needs a set of tools for such tasks as capturing, modifying, editing, sequencing, synchronizing, archiving, versioning, and backing up. The heart of this set of tools is an Integrator which can be used by the developer to assemble various multimedia objects into an IMMA and provide the end-user with ways of interacting with that IMMA.

This work focuses on that essential tool — the multimedia Integrator. The Integrator uses a graphical interface which exploits a notation like that of a musical score. The IMMA produced by the Integrator is inherently parallel and includes elements required to define the asynchronous nature of applications, along with a standard set of multimedia objects. The timing and synchronization representation used by the Integrator is based on the model presented by the HyTime standard group. We also have explored the representation of our objects and their attributes according to the Multimedia Hypermedia Expert Group (MHEG) standard model as defined by the MHEG committee.

This work formalizes the design of the Integrator and its constructs using the HyTime Draft International Standard (ISO/IEC DIS 10744). Multimedia application objects and their attributes are described, timing and synchronization aspects of the Integrator and some examples of IMMAs are explained, and several of these examples using the HyTime document structuring language are given.
To my parents,

Prema Narasimhan and V.K. Narasimhan
ACKNOWLEDGEMENTS

I would like to thank Dr. Edward A. Fox for his valuable support, guidance, understanding, and encouragement during the research and writing of this document. His comments and criticisms were very constructive and helpful. I would also like to thank Dr. Deborah S. Hix and Mr. Edward E. Schwartz for serving on my committee. They have been a source of many fruitful discussions and their constant willingness to help is appreciated. Thanks go to Dr. Steven Newcomb, Mr. Neill Kipp and Ms. Victoria Newcomb at Technoteacher, Florida, and to Mr. Erik Naggum at Naggum Software, Norway, for their valuable pointers regarding HyTime and SGML. I would also like to thank Dr. Antonio Siochi, for his ideas on the Integrator, and NCR Corporation, Atlanta, for funding this work.

Working on this project has been enjoyable experience due to a fun group of people, the “Foxlettes”; Ghaleb Abdulla, Sangita Betrabet, Siva Challa, Qifan Chen, Prabhakar Koushik and William Wake have lent support, good humor and discussions on various aspects of life. Vish Ramachandran, Vasant Sanjeevan and Mahesh Ursekar have been a good source of friendship and I thank them for all the fun weekends and rides to and from Roanoke.

Most importantly, I would like to thank my uncle, Kalyanaraman Rangarajan, and my brother, Ajit Narasimhan, for always being there to put up with me under all circumstances. I am deeply indebted to my parents, Prema Narasimhan and V.K. Narasimhan, for the unconditional sacrifice, support, and encouragement throughout my life. To them, I owe everything.
# TABLE OF CONTENTS

1 Introduction  
  1.1 Overall Scope ........................................... 2  
  1.2 Background ............................................... 3  
  1.3 Problem Statement ........................................ 4  
  1.4 Outline of Thesis ......................................... 4  

2 Literature Review  
  2.1 Multimedia Field .......................................... 6  
  2.2 Standard Generalized Markup Language - SGML .............. 10  
  2.3 HyTime and MHEG Standards ................................ 11  
    2.3.1 HyTime ............................................. 11  
    2.3.2 MHEG ............................................... 12  

3 Our Previous Work  
  3.1 Digital Video Interactive - DVI .......................... 13  
  3.2 Interactive (Digital) Multimedia (Applications) Development Environment - IMDE ................................. 15  
  3.3 The Integrator ............................................. 17  
    3.3.1 The Basic Metaphor .................................. 17  
    3.3.2 Language Constructs .................................. 19  
    3.3.3 Using the Integrator ................................ 22
CONTENTS

3.3.4 Version 1 Prototype ................................................. 22
3.3.5 Version 2 Prototype ................................................. 23

4 HyTime Description of Integrator ................................. 24
  4.1 HyTime Objects and their Attributes ............................ 24
    4.1.1 Listing of Available Multimedia Objects .................. 25
    4.1.2 Hierarchy of Multimedia Application Objects  ............. 26
  4.2 Synchronization and Timing Aspects in the Integrator ....... 29
    4.2.1 Temporal Arrangements of a Pair of Events ............... 29
    4.2.2 Some Rules for Placement of Multimedia Objects in Time and Space 29
    4.2.3 Types of Synchronization in the Integrator ............... 33
    4.2.4 Other Issues Regarding Synchronization ................... 38
    4.2.5 Conversion of VTU to RTU .................................. 43
    4.2.6 Controlling of Batons .................................... 45
    4.2.7 Other Issues and Solutions ................................. 47
  4.3 Mapping Representations: Integrator to HyTime ............... 47
    4.3.1 Mapping Composite Object to HyTime Notation ............. 48
    4.3.2 Mapping Computation Object to HyTime Notation .......... 51
    4.3.3 Mapping Conditional Object to HyTime Notation ........... 52
    4.3.4 Mapping Link (Hyper) Object to HyTime Notation ......... 54
    4.3.5 Mapping Loop/Iteration Object to HyTime Notation ....... 55
    4.3.6 Mapping Usable Objects and their Attributes to HyTime Notation 57
    4.3.7 Mapping Orelation (Run-time step) Attribute to HyTime Notation 58
    4.3.8 Mapping Score Construct to HyTime Notation ............. 60
    4.3.9 Mapping Synchro Object to HyTime Notation .............. 61
    4.3.10 Mapping Tie Object to HyTime Notation .................. 62
    4.3.11 Mapping Top Level TimeLine and Time for Tracks to HyTime Notation 63
    4.3.12 Mapping Tracks Construct to HyTime Notation ........... 64
CONTENTS

5 Conclusions and Future Work 66
  5.1 Conclusions ................................................................. 66
  5.2 Future Work ................................................................. 67

A HyTime Listing of Objects 79

B HyTime Listing of Examples 111
  B.1 Slideshow with Audio ....................................................... 112
     B.1.1 User View of the Application ........................................ 113
     B.1.2 HyTime Representation ............................................. 114
  B.2 Slideshow with User Control ........................................... 116
     B.2.1 User View of the Application ........................................ 117
     B.2.2 HyTime Representation ............................................. 118
  B.3 Demo Program .................................................................. 120
     B.3.1 User View of the Application ........................................ 124
     B.3.2 HyTime Representation ............................................. 125
  B.4 DTD for the IMMA Examples ........................................... 130

C Integrator Meta-DTD 139

D Glossary 141
LIST OF FIGURES

3.1 Score of the Integrator .................................. 18
3.2 Dialogue Box to Edit Multimedia Object - Still ........ 20
3.3 Integrator Graphical Interface .......................... 21
3.4 Hierarchy of Multimedia Application Objects .......... 30
3.5 Temporal Ordering of Two Events ....................... 31
3.6 Illustrations Regarding Some Rules for Placement of Multimedia Objects in Time and Space .................... 34
3.7 Independent and Interruptible Synchronization of Objects .... 36
3.8 Simple Situations Concerning Synchronization ............ 39
3.9 Complex Situation Concerning Synchronization .......... 41
3.10 Using Baton to Project Time from One FCS to Another .... 44
3.11 Defining Time Value for Composite Objects ............... 46
3.12 Integrator Architecture ................................. 49
3.13 Flow of Execution through Scores Containing Composite Objects .......... 50

B.1 Integrator Screen Showing Slideshow with Audio .......... 112
B.2 Pictorial Description of Slideshow with Audio ............ 114
B.3 Integrator Screen Showing Slideshow with User Control .... 117
B.4 Integrator Screen Showing Demo Program .................. 121
B.5 Integrator Screen Showing Slideshow Composite .......... 121
B.6 Integrator Screen Showing Quiz Composite ................. 122
LIST OF TABLES

B.1 Elements and Keywords Used in the Examples and DTD .............. 137
B.2 Elements and Keywords Used in the Examples and DTD (contd.) .... 138
Chapter 1

Introduction

In order to produce an Interactive MultiMedia Application (IMMA), the developer of that application needs a set of tools for capturing, modifying, editing, sequencing, synchronizing, archiving, versioning, backing up, etc. The heart of this set of tools is an Integrator which can be used by the developer to assemble various multimedia objects into an IMMA and provide the end-user with ways of interacting with that IMMA.

This work focuses on that essential tool — the multimedia Integrator. The Integrator uses a graphical interface which exploits a notation like that of a musical score. The IMMA produced by the Integrator is inherently parallel and includes elements required to define the asynchronous nature of applications, along with a standard set of multimedia objects. The timing and synchronization representation used by the Integrator is based on the model presented by the HyTime standard group. We also have explored the representation of our objects and their attributes according to the Multimedia Hypermedia Expert Group (MHEG) standard model as defined by the MHEG committee.

This work formalizes the design of the Integrator and its constructs using the HyTime Draft International Standard (ISO/IEC DIS 10744). We list the various multimedia application objects and their attributes, explain the timing and synchronization aspects of the Integrator and some examples of IMMAs, and encode several of those examples using the
CHAPTER 1. INTRODUCTION

HyTime document structuring language.

1.1 Overall Scope

This thesis formalizes design of an IMMA authoring system called the "Integrator" [114] using the HyTime Draft International Standard [52,55,68]. The HyTime standard is an application of the International Standard Organization standard ISO 8879, Standard Generalized Markup Language (SGML), and addresses issues regarding standardizing multimedia applications and exchange of documents between different computer systems [54,68,100].

This work involves defining a set of objects and constructs that are necessary to represent an IMMA, listing different timing and synchronization issues which occur when defining IMMAs, and mapping IMMAs from the Integrator notation to the HyTime notation. It shows through the use of examples how IMMAs can be developed, represented using a standard intermediate format, and finally transformed to an executable file which can be executed to show the final presentation. The examples chosen (listed in the Appendix) incorporate most of the multimedia application objects and illustrate important timing and synchronization issues. The multimedia object list is not by any means a complete set, but has been designed to be easily extended to allow different platforms to support the Integrator.

This work uses those modules of HyTime and SGML that would affect issues related to the design of the Integrator. There are some aspects of these standards that have been ignored, as these standards have been defined for very general applications. This work involves minimal implementation - thus simple prototypes of the front-end of the Integrator have been built using authoring languages like HyperCard on the Macintosh system and MEDIAscript on Intel's Digital Video Interactive (PRO 750) system.
CHAPTER 1. INTRODUCTION

1.2 Background

Technical work on important standards such as Joint Photographic Experts Group (JPEG) [126], Motion Picture Experts Group (MPEG) [75], Hypermedia/Time-Based Structuring Language (HyTime) [52], Standard Generalized Markup Language (SGML) [54], Standard Music Description Language (SMDL) [53], etc., will enable users to interchange files with multimedia objects and display them on a variety of different computers with completely different architectures. Progress in the areas of algorithms, software, and hardware have resulted in systems with complex architectures which have capabilities such as compression and decompression of audio, video, and still images, intensive graphics processing, and digital signal processing [42,40]. All of these contribute to the emergence of digital multimedia technology. This technology has resulted in new and creative ways of learning where the user can interact with the multimedia system and take part in the education process rather than just be a passive listener [77]. Systems such as Palenque [128] and the Advanced Learning Technology’s Code Inspection software from the Software Engineering Institute at Carnegie Mellon University [121] use the methodology of discovery based learning where the user learns by exploring. Desktop multimedia publishing [43], mail enhanced with annotations [18], multimedia training/education/entertainment, computing with windows for motion video, and advanced conferencing systems are among the most important classes of applications, fitting in with new proposed reference models for hypermedia and multimedia [37]. Developments in interactive videodiscs have made it possible to work with analog audio and video on computer systems. Digital storage media, including optical discs and magnetic disks that can be networked, have also helped digital multimedia to be shared, preserved, and distributed [45]. Since media such as audio and video consume large amounts of space and bandwidth, techniques such as compression and decompression are of particular importance [122]. Two well known types of systems of this class are CD-I, Compact Disc Interactive, developed by Philips along with Sony [103,25], and DVI, Digital Video Interactive, now involving Intel, IBM, and others [81]. Application development tools
CHAPTER 1. INTRODUCTION

called "authoring languages" have been developed and are gradually improving, and can go beyond those designed around the capabilities of videodisc systems. This work with the Integrator involves designing one such authoring system that can be used to build IMMAs on multimedia systems such as those described above.

1.3 Problem Statement

This work deals with the formalization of the design of the Integrator and its constructs used in building Interactive MultiMedia Applications (IMMA). It also gives solutions to some problems such as:

- classification of multimedia application objects and the definition of their attributes (Section 3.4),

- timing of multimedia objects in an IMMA (Section 3.5), and

- issues of synchronization of multimedia objects in an IMMA (Section 3.5).

1.4 Outline of Thesis

Chapter 1 gives an introduction to this thesis by defining the overall scope of the thesis, giving relevant background information about the area of this work and the standards involved, and defining the problem to be solved.

Chapter 2 is a comprehensive literature review of the history, trends, and future of the multimedia area, the Standard Generalized Markup Language (SGML), the Hypermedia Time-based Document Structuring Language (HyTime), and the Multimedia Hypermedia Experts Group (MHEG).

Chapter 3 describes the previous work done at Virginia Tech related to this thesis. It describes work related to Digital Video Interactive (DVI), Interactive Multimedia Development Environments (IMDE), and the versions of the Integrator prototype that were built
CHAPTER 1. INTRODUCTION

on the Macintosh platform using the HyperCard authoring language and the MEDIAscript authoring language, on the Intel PRO 750 DVI platform.

Chapter 4 is the core of this thesis, where the HyTime description of the Integrator is defined. The hierarchy of the multimedia application objects (and their attributes) is defined and a HyTime listing of the same is included in the Appendix. This chapter also describes timing and synchronization problems, and gives solutions to solve these issues through the use of examples. Mapping of the objects and the constructs from the Integrator notation to the HyTime notation is also discussed in this chapter.

Chapter 5 lists the conclusions drawn from this work and describes the future work that could be done related to this thesis.
Chapter 2

Literature Review

For the purpose of literature review, recent advances made in the field of multimedia (related to this work) can be broadly grouped into three categories:

- Multimedia Field
- Standard Generalized Markup Language (SGML)
- HyTime and MHEG Standards

2.1 Multimedia Field

Almost every area of computer science has a role to play in this high growth field. This field has far reaching effects as a multibillion dollar industry expanding the scope of computers and communication systems further into the worlds of publishing and television [40]. This field integrates several media types such as text, graphics, animation, audio, and video into a single computer document. Stefanac and Weiman list some of the multimedia production tools that are used commonly with Macintosh PCs [120]. In their book, Ambron and Hooper of Apple Computer Corporation provide an overview of what multimedia is and how it can be used in education [5]. In a short article, McCarthy discusses the advantages and disadvantages of this area, citing examples of school students using multimedia concepts.
CHAPTER 2. LITERATURE REVIEW

in their class assignments [90]. Bergman and Moore have investigated the various steps involved in producing a multimedia project and have discussed the various stages in detail [15].

Storage plays a very important part in multimedia systems considering the amount of data that has to be transferred between the storage units and the system to present a continuous flow of output for the multimedia application. Improvements in optical storage including CD-ROM [113] and high capacity magnetic disks have made this possible [46]. Hoagland describes the storage revolution that has taken place in the last few years and mentions the changes in the different ways in which people store data [65]. Lambert and Ropiequet, and Ropiequet, Einberger and Zoellick have compiled a classic collection of articles in their two volume set [74,109], which deals with the various aspects of CD-ROMs starting from the preparation of data [1] to the publishing and distribution of CD-ROM discs [74,109]. In a survey paper, Fox gives a detailed review of the publishing and access issues for optical discs and CD-ROMs [48]. Systems like CODER [39] and LEND [38] developed and tested at Virginia Tech can integrate search and retrieval techniques with hypertext facilities and use SGML and a lexicon for their database [47]. Laserdisc technology also plays an important part in this field and aims to cover a different set of multimedia applications [112]. Mageau gives a brief description of this technology and lists some of the applications developed [87].

Applications that allow linking of multimedia objects from one document to another are classified as hypermedia applications. The term “hypermedia” was coined by Nelson. There are several groups of people within the industry who are trying to standardize hypermedia systems and applications so that users can then interchange information contained in hypermedia documents. Several articles on standardization can be found in the proceedings of a workshop held by the National Institute of Standards and Technology (NIST) [92]. Conklin in his articles has provided a detailed survey of the concepts that are used in hypertext systems and has also included some of the classic hypertext systems including Bush’s Memex, Englebart’s Augment, Nelson’s Xanadu Project, and Trigg’s Textnet
CHAPTER 2. LITERATURE REVIEW

[30,31]. Trigg has investigated and reported on a network-based approach to text handling for the Online Scientific Community [125]. Some of the problems faced by hypertext system users are outlined by Nielsen through the use of his system developed using the HyperCard software [95].

Another area that plays an important part in multimedia systems is Human Computer Interaction. Research has been done in this area to develop electronic documents and user interfaces that can be used easily and effectively by users with different backgrounds [89]. To break the language barrier and to improve communications between users, command line processing has been replaced by graphical user interfaces incorporating icons to define several common notations [89]. Several tools have been developed to make it easier for programmers to build multimedia systems. Work to evaluate these tools for consistency, functionality, flexibility and level of productivity has been performed [63,64,86]. Authoring languages like HyperCard [9], Authology Multimedia [24], and MEDIAscript [82], can be supported by paint programs like Lumena [123], and are available to easily and rapidly build prototypes of systems. The multimedia market is being flooded by systems that have different capabilities to integrate multimedia objects into applications built for different classes of users.

Multimedia Systems and Techniques

Almost all multimedia systems emphasize one main objective - communication. Multimedia systems such as MediaView built on the NeXT platform can be used for digital publication and are designed on a WYSIWYG (What You See Is What You Get) metaphor [104, 105]. The Integrator (which will be dealt with in detail in the subsequent chapters) is designed with a graphical interface and stresses the importance of how developers can build IMMAs rapidly [114]. Several systems such as Intermedia [59,91], Apple’s QuickTime [10], Commodore’s Amiga Vision [29], and Stanford University’s MAEstro [36], are based on a time-line metaphor where the system keeps track of processes that are being executed at a particular point in time. Systems such as EVA (Experimental Video Annotator) [84,85]
CHAPTER 2. LITERATURE REVIEW

and Pygmalion [85] built at MIT describe multimedia tools for recording multimedia events, annotating video data, shared editing control, interactive video viewing and editing, video data analysis, multimedia message system, etc. that can enhance the building of multimedia applications.

Multimedia systems can be used to store and retrieve information that is not in the text format. One such system that is based on videodisc, called CLORIS, can be used to describe, retrieve, and discuss video stills and sequences [102]. Systems such as MULTOS [16] and MINOS [27] can also be used for query processing, document presentation, information extraction, and document formation. Data used in the above systems vary from text to graphics to images including audio [8,20]. Chang and Leung demonstrated a knowledge-based message management system that uses message filters to filter out junk messages [26]. Knowledge representation techniques have been applied to some systems such as the Prototype Electronic Encyclopedia for improving access to knowledge resources [127]. Thus we see a trend from traditional documents that used only text and pictures or graphics to current documents that contain still images, audio and even video. The Electronic Newstand built by Donath presents a design of an intelligent interface to a variety of news sources in several media [35]. Other multimedia systems that are based on CD-ROMs such as DVI [44] and CD-I [101] also use compression and real-time decompression techniques to store and display audio and video.

There are many techniques that are used for image, video and audio compression and decompression [61]. Committees such as MPEG for video and audio compression [75] and JPEG for still images and video compression [126,28] are working on the theories, algorithms, modes of use, and classes of applications. In a short article on the px64 standard Liou explains how motion video can be transmitted over digital networks [80]. Compression is very useful for both data storage and data transmission [40]. Bell, Cleary and Witton have compiled a number of different techniques for text compression in their text [14]. Techniques such as Vector Quantization [11,12,57], DCT [107], Prediction [78], Fractals [13], Digitization [78] and Sub-band coding [79,23] are also useful in compression of mul-
CHAPTER 2. LITERATURE REVIEW

timedia data. Rao and Yip's text on DCT covers applications, theories, algorithms, and other related work [107]. The MBASIC system, an image coding method, utilizes a 3-D model of an object that is to be reproduced [3]. Several different techniques for compression and decompression have also been discussed in the Scientific Data Workshop held at NASA [106]. Compression and decompression techniques for audio have been dealt with in [93,6]. Although many techniques are still to be perfected, most of them perform the required functions to some satisfaction depending on the applications involved.

2.2 Standard Generalized Markup Language - SGML

SGML is the International Organization for Standardization (ISO) standard for document description (ISO 8879) which is designed specifically to enable text interchange. By far, the most comprehensive document on SGML has been the text by Goldfarb in which he has presented a tutorial, overview and listed an annotated version of the ISO 8879 document [54]. Other books on SGML by Bryan [21] and Herwijnen [62] are useful to those who are novice SGML users. Bryan's book, one of the first SGML books to be published, gives a brief description of the constructs used in the SGML language. Herwijnen's book deals with an application of SGML, CALS [116], and describes the constructs of SGML through the use of CALS. Smith and Stutely have drawn up a list of all the constructs used in the ISO 8879 and provided a detailed index [117]. SGML defines just the structure of the document being coded. Another ISO standard, called the Document Style Semantics and Specification Language (DSSSL), addresses the problem of describing the layout independently of formatting systems or processors [2].

Structured documents that describe a collection of objects at various levels (higher level objects formed from more primitive objects) represent logical relationships between components of the document. The principles underlying the logical structuring of documents and the advantages that can be drawn from it are listed in the collection of survey articles in [7]. Salton's book deals with the handling of texts using automatic equipment and text process-
CHAPTER 2. LITERATURE REVIEW

Articles written by DeRose [33] and Coombs et al. [32] describe the different types of document markup. These articles also deal with the description of compound documents, representing objects based on their content, and representing objects at different levels of granularity. Smith has investigated the history of SGML in her article and has listed some of the major applications of SGML [116]. The technical contents of an SGML document, as specified by its Document Type Definition (DTD) and declaration constructs, are defined in [119]. This booklet also deals with simple and practical examples which can be easily understood by SGML beginners. The document by Software Exoterica Corporation describes the syntax of a SGML Declaration and lists a template Declaration that can be used by SGML users [118]. Finally the SGML SIGhyper (Special Interest Group on Hypertext and Multimedia) group keeps track of the advances in this field and publishes a newsletter that covers different activities, events and efforts to make SGML usable by everyone [111].

2.3 HyTime and MHEG Standards

2.3.1 HyTime

The HyTime Document Structuring Language is defined in ISO/IEC DIS 10744 (Draft International Standard) which has gone into its last phase of voting, due to become a standard in April 1992 [68]. The article by Newcomb et al., a good start for first time HyTime readers, gives an introduction to the HyTime standard and describes the different modules of HyTime [98]. Also, an electronic version (and a related hardcopy version [99]) gives an overview of the standard by using simulation of some real-life examples to demonstrate the mapping of virtual time to real time and rendition of different processes. Some of the concepts regarding synchronization of two or more events can be found in [4,115]. Sloan gives a good insight to the synchronization and rendition modules of HyTime through the use of real-life situations [115]. Newcomb’s short articles present examples of the use of batons and translation of virtual or music time to real time [96,97].
CHAPTER 2. LITERATURE REVIEW

The SGML SIGhyper formed in 1991, puts out a newsletter [111] that describes the latest developments, activities, events, and other features related to the HyTime standard. It also lists several examples of HyTime documents that the reader could refer to while creating his/her own documents. An example of a HyTime document is also listed in [70]. Johnson has suggested applications for HyTime that are not based on music in [69].

The Standard Music Description Language (SMDL), defined in ISO/IEC CD 10743, is an application of HyTime that defines a language for the representation of music information, either alone or in conjunction with text, graphics, or other information needed for publishing or business purposes [53]. A document formatter for the SMDL has been developed by Kipp that takes as input a text document written in SMDL and outputs a text file command sequence for either a computer based synthesizer or a musical notation generator [71].

2.3.2 MHEG

The working group WG12, of the sub-committee SC2, of the ISO/IEC JTC1, called the Multimedia/Hypermedia Experts Group (or MHEG), works towards the representation of synchronized hypermedia and multimedia objects [73]. They have taken an object oriented approach to standardize the common multimedia and hypermedia objects. Coded representations of hypermedia and multimedia information can be found in the technical documents [56,73]. Markey presents the emerging hypermedia standards MHEG and HyTime and outlines through the uses of several examples how the current hypermedia market place prepares for these standards [88]. Hovey addresses the framework for hypermedia and multimedia standardization and lists recommendations for the use of these standards [66]. The coded representation of synchronized multimedia objects, coded representation of hypermedia objects, representation and coding principles, and the field of application are described in the MHEG working document, version 4 [17].
Chapter 3

Our Previous Work

The multimedia project at Virginia Tech, co-funded by NCR Corporation and the Virginia Center for Innovative Technology, revolves around a unique combination of theoretical and practical work in the area of interactive digital video. We have built upon earlier work – both our own and that of others – in information storage and retrieval, CD-ROM publishing, interactive videodisc applications development, and human-computer interaction. This project was launched when Virginia Tech was chosen by Intel as a beta test site in 1988. This chapter describes some of the previous work done in the areas of Digital Video Interactive, Interactive (Digital) Multimedia (Applications) Development Environment - IMDE, and the Integrator applications development tool.

3.1 Digital Video Interactive - DVI

Digital Video Interactive is a technology from Intel Corporation for developing and presenting multimedia applications in a completely digital format [81], [108] and [83]. Such multimedia applications may combine audio, still images, and motion video with text, graphics, and animation, and most importantly allow integration and real-time manipulation of any of these by a user [19] and [67]. The DVI system software is available currently on PC/AT or PS/2 platforms running MS-DOS [22] and [72]. The older version of the system software
was based on the metaphor of a VCR player, and the newer version, the Audio Visual Ker-
nel (AVK), is based on the metaphor of a production studio [58]. DVI technology allows
integration of full-motion digital video, computer-generated and/or digitized graphics and
text, and audio information utilizing a compact disc storage medium with 650 megabytes
of capacity as well as conventional magnetic disks [59], [124] and [129]. Capture and play-
back of all multimedia components are supported by proprietary compression and real-time
decompression techniques [76]. This technology has brought closer several other fields of
computer science including artificial intelligence, databases, information storage and re-
trieval, and simulation [51,94]. Several examples of interactive multimedia applications are
listed in [34] and [49].

Our previous research identified an urgent need for flexible high-level interactive tools
in a comprehensive environment to support development of multimedia applications, and
we have since produced some prototype tools. Such tools will improve the productivity of
multimedia application developers, thereby significantly reducing project costs. We have
also interviewed other application developers to obtain information and descriptions of their
experiences and impressions of this field. We have investigated the assets and liabilities of
DVI technology, comparing DVI and related technologies such as IVD and CD-I. To provide
theoretical foundations for our practical investigations we have produced a taxonomy of the
components of a multimedia application. To establish a baseline for tools to support DVI
development we have analyzed capabilities of available "authoring" and other development
tools. We have also identified the basic requirements - functionality, usability, and others
- of an interactive environment (tool) for developing multimedia applications [41]. These
requirements are listed in the next section. We have developed prototype applications such
as the Product Theater and the Architecture Image Database [41], to test capabilities of
this technology. The Virginia Tech multimedia project hopes to break new ground in how
interactive software systems are developed and used.
CHAPTER 3. OUR PREVIOUS WORK

3.2 Interactive (Digital) Multimedia (Applications) Development Environment - IMDE

The research team at Virginia Tech, working to advance the science, engineering, and art of multimedia information systems, has studied such questions as:

- What is the nature of a multimedia application?
- What are the different areas for such applications?
- How can multimedia applications be developed in a time-efficient, cost-effective manner?

To address such questions we built a taxonomy of the components of an interactive digital video application and explored other such interactive video applications. We also investigated existing tools for developing interactive systems to determine the needs of multimedia application developers. We built tools to speed up our work and aid other developers. Our goal is to have an environment for rapid prototyping and construction of multimedia systems, allowing flexible user access [41].

To lower the costs of building applications, we focused on IMDE. The general requirement for such an environment was to have a seamless integration of such tools that are also independent of multimedia technology. The set of tools should be modular and support most of the common functions required by the developers. Some usability requirements for such tools are: time to perform tasks with the tools must be less than that required with conventional techniques, developer must have a direct manipulation interface to tools whenever possible, windows should be available for different tasks, and there should be context-sensitive help. Some of the functional requirements include [41]:

- capture of multimedia data,
- modification/editing of multimedia data,
CHAPTER 3. OUR PREVIOUS WORK

- synthesis/editing of computer-based data,
- combination of captured data with computer-generated data,
- sequencing and synchronizing of multimedia data,
- specifying and supporting user interaction,
- design,
- evaluation,
- simulating multimedia delivery systems,
- soft-copy and hard-copy output, and
- archiving, backup, versioning and recovery.

A basic set of tools was identified for developers using an interactive multimedia application development/utilization environment [40] including tools for:

- multimedia data capture,
- multimedia data editing/synthesizing (for animation, audio, drawing, music, painting, and video),
- multimedia scripting,
- multimedia data integration and sequencing,
- multimedia database layout and storage/retrieval,
- user interface development,
- simulation, testing, publishing assistance,
- archiving, backup, versioning, recovery,
CHAPTER 3. OUR PREVIOUS WORK

- project management, and
- run-time support for application use.

After building a tool for identifying hotspots or buttons, we designed and began prototyping the Integrator as the most crucial component in the environment. The Integrator, considered the central tool in the environment, is designed to be platform-independent, and can be used to build simple interactive multimedia applications. When all the raw materials for an multimedia application are assembled, the Integrator can support putting the interactive program together. A designer can use the Integrator's graphical interface which supports iconic representation of all the common type of digital multimedia objects to specify ordering, control and other relationships. When the description of the application is complete, it is translated into a program that can then be run on multimedia systems such as the DVI platform.

3.3 The Integrator

This section is a summary of [114] which deals with the Integrator in detail.

3.3.1 The Basic Metaphor

The Integrator can be used by an IMMA developer to assemble various multimedia objects into an IMMA, and provide the end-user with ways to interact with that IMMA. (Here we distinguish between the developer as the person who assembles the multimedia objects to form the IMMA, and the end-user as the person who actually interacts with and uses the IMMA.) The Integrator is based on the metaphor of a musical score sheet. An IMMA is represented on a set of horizontal tracks, analogous to the staves in a score. This is illustrated in Figure 3.1a, which is based on the second version of our prototype Integrator.

Multimedia objects are placed on tracks just like notes placed on the staff in a score. Timing and synchronization of multimedia objects are determined by horizontal and vertical
Figure 3.1: Score of the Integrator
CHAPTER 3. OUR PREVIOUS WORK

relationships to other multimedia objects. The score in Figure 3.1a is an example of a simple slideshow with three still images on one track and an audio object on the other track. Objects are placed on the tracks by dragging them from the menu bar and then releasing them at the required spot. Attributes for all chosen objects can be entered or modified through a dialogue box interface as shown in Figure 3.2.

Numerous other objects like multimedia objects (e.g., still images, video, composites), user input (e.g., get event, get mouse click, get keystroke), control (e.g., branch, iterate), and transitions (e.g., fade, wipe right, cut) can be used in an IMMA. Each component of an IMMA is considered as an object and is defined in terms of associated attributes. An example of the attributes associated with the still image object is shown in Figure 3.2b.

3.3.2 Language Constructs

The Integrator is essentially an interpreter and compiler for the graphical programming language, illustrated in Figure 3.3, that is based upon our elaboration of the musical score metaphor.

Some of the tasks supported by the Integrator include the following [114]:

- sequencing and synchronizing multimedia objects,
- determining iteration and looping sequences among objects,
- determining branching on conditions between objects,
- accessing external computation routines,
- defining user input,
- defining transitions among images, and
- defining multimedia data.
CHAPTER 3. OUR PREVIOUS WORK

(a)

Object - Still Image

Edit or Delete this object?

Edit  Delete  Cancel

(b)

Edit - Still Image

<table>
<thead>
<tr>
<th>Name</th>
<th>snoopy</th>
</tr>
</thead>
<tbody>
<tr>
<td>File</td>
<td>samp256.c16</td>
</tr>
<tr>
<td>Resolution</td>
<td>512 x 480</td>
</tr>
<tr>
<td>X, Y pos</td>
<td>0, 0</td>
</tr>
<tr>
<td>Duration</td>
<td>15 secs</td>
</tr>
<tr>
<td>Format</td>
<td>compressed</td>
</tr>
<tr>
<td>Link to</td>
<td>none</td>
</tr>
<tr>
<td>Hot spots</td>
<td>none</td>
</tr>
</tbody>
</table>

Figure 3.2: Dialogue Box to Edit Multimedia Object - Still
CHAPTER 3. OUR PREVIOUS WORK

<table>
<thead>
<tr>
<th>Still Image</th>
<th>Transition - Still Image</th>
</tr>
</thead>
<tbody>
<tr>
<td>Video Segment</td>
<td>Transition - Video</td>
</tr>
<tr>
<td>Audio</td>
<td>Transition - Audio</td>
</tr>
<tr>
<td>Composite image</td>
<td>Timer</td>
</tr>
<tr>
<td>Escape to C code function/ Computation</td>
<td>User Input - Keyboard</td>
</tr>
<tr>
<td>Conditional Function</td>
<td>User Input - Mouse</td>
</tr>
<tr>
<td>Loop/Iteration</td>
<td>Help</td>
</tr>
</tbody>
</table>

Figure 3.3: Integrator Graphical Interface
CHAPTER 3. OUR PREVIOUS WORK

3.3.3 Using the Integrator

Some of the functions performed by the Integrator can be invoked using the pull-down menu associated with the Options entry in the menu bar (see Figure 3.1b). Tie Objects can be used to synchronize multimedia objects that have been placed on different tracks of a score. The user can also select any multimedia object that has been placed on the tracks, and view (still image or video) or hear (audio) the segment associated with that object. This is accomplished by selecting the Present Object choice and then the desired object. Composite Object gives the Integrator the ability to manage hierarchically structured IMMAs. By editing a composite, the IMMA developer automatically goes down one level in the structure. To get back to a higher level the developer can use the Previous Level entry in the Options menu.

When the design of an IMMA is complete, an executable representation of the application can be produced by selecting the Execute option. This automatically invokes a back-end interpreter to 'execute' the IMMA. The interpreter can also be run in a stand-alone mode, i.e., invoked at the operating-system level without involving the Integrator.

The Integrator is usable by more than just programmers due to the point-and-click graphical interface. It also allows for development of IMMAs that have user interaction. It supports synchronization and temporal relationships across multiple tracks and produces an executable IMMA, with little or no programming.

3.3.4 Version 1 Prototype

Our initial prototype can be run on any type of Macintosh running HyperCard connected by a serial line to the DVI platform. Since multimedia applications are event- and time-based like musical performances, we exploit the metaphor of a score. Horizontal tracks correspond to logical input and output devices like windows or audio channels. Icons representing media objects like stills, video, audio, etc. can be selected and then connected with various transitions (special effects) and control elements.
CHAPTER 3. OUR PREVIOUS WORK

When changes are made to the IMMA, the final specification can be analyzed and translated to our specialized intermediate representation language. The representation was system dependent since the IMMA support was limited to just a DVI slideshow application. The representation was formed as an ASCII file which collected information in specific fields on the names of still images, type of transition effects, time for the still images to be displayed on the screen and the time for the transition effect. When this file was sent to the DVI system over a serial line connection the data file was read by a controlling program and the application was executed displaying the slideshow application.

When fully implemented on any multimedia system, the Integrator should greatly simplify the development of interactive multimedia applications involving high quality audio, video, images and graphics.

3.3.5 Version 2 Prototype

The second version of the Integrator was built on the DVI system using MEDIAscript Version 1.5, an authoring language. The set of multimedia objects was enhanced from just still images and transitions in the first version, to include other multimedia objects such as audio, video, computations, composites, and input. Different attributes for all of the multimedia objects were also formed and formalized using the HyTime standard. This version was mainly built to demonstrate the capabilities of the front end of the Integrator. Using this front end, the developer could select the required multimedia objects and assemble them into an IMMA. The attributes for the selected objects could be set by entering the edit mode whereby the developer could change the required values. An intermediate representation of applications could also be generated with this version. This representation followed the same design criteria as the previous version but contained more extensive information about the IMMA. It allowed for some functions like deleting objects when once placed on a track, copying attributes from one object to another without creating a new object, and specifying sets of defaults values for different objects and filling them in the appropriate fields when these attributes were not given.
Chapter 4

HyTime Description of Integrator

This chapter formalizes the objects and the constructs used in the Integrator using the HyTime standard. Attributes for the available objects are listed in HyTime and the timing and synchronization issues are discussed and solutions to some of them are discussed. The last section explains the process of mapping IMMAs from the Integrator to the HyTime notation to represent applications in a standardized manner.

4.1 HyTime Objects and their Attributes

This section describes multimedia application objects and their functions in an IMMA. This set of objects was formed by considering examples from the class of applications that has a linear document structure. This means that the structure of the application can be represented as a document. Note however, the execution of the application need not be linear. The execution would depend on the end-user interaction with the application.

A minimal set of objects was formed such that applications with moderate complexity could be defined using this set. These objects have been organized in a logical hierarchical manner and grouped into two categories depending on the way they are used in the Integrator. The first category is the “Usable” multimedia objects - those objects that are directly incorporated in the IMMA. Objects such as StillImage, Video, Audio, Transition,
CHAPTER 4. HYTIME DESCRIPTION OF INTEGRATOR

Conditional, Composite, Input, and Baton are typical of this category. The second category is the “Supportive” multimedia objects - those objects that support definition and building of IMMAs and are not themselves incorporated in the iMMA. Objects such as Execute, PreviousLevel, PresentObject, Pool, HotSpot, and Window are typical of this category.

Another way of categorizing multimedia objects is into four classes based on the functions and methods of the objects. This hierarchy is influenced by the hierarchy listed in the MHEG document [73]. A formalized listing of all the usable and supportive multimedia objects is given in the Appendix.

4.1.1 Listing of Available Multimedia Objects

Following are brief explanations of the multimedia objects found in the Integrator. These are categorized into usable or supportive types.

<table>
<thead>
<tr>
<th>Object</th>
<th>SubClass</th>
<th>Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Audio</td>
<td>–</td>
<td>Usable</td>
<td>Audio sequence</td>
</tr>
<tr>
<td>Animation</td>
<td>–</td>
<td>Usable</td>
<td>Animation sequence</td>
</tr>
<tr>
<td>Baton</td>
<td>Master</td>
<td>Usable</td>
<td>Master (Main) baton for appln. (highest level)</td>
</tr>
<tr>
<td></td>
<td>Slave</td>
<td>Usable</td>
<td>Baton for score or track (lower level)</td>
</tr>
<tr>
<td>Composite</td>
<td>–</td>
<td>Usable</td>
<td>Composite object, composed of other objects</td>
</tr>
<tr>
<td>Computation</td>
<td>–</td>
<td>Usable</td>
<td>External computation</td>
</tr>
<tr>
<td>Conditional</td>
<td>Do While</td>
<td>Usable</td>
<td>Do (Action) While (Condition)</td>
</tr>
<tr>
<td></td>
<td>For Loop</td>
<td>Usable</td>
<td>For (expr:expr:expr) (Actions)</td>
</tr>
<tr>
<td></td>
<td>If Then Else</td>
<td>Usable</td>
<td>If (condition) Then (action1) Else (action2)</td>
</tr>
<tr>
<td></td>
<td>Loop N</td>
<td>Usable</td>
<td>Loop N times</td>
</tr>
<tr>
<td></td>
<td>On X Do</td>
<td>Usable</td>
<td>On (X) Do (action)</td>
</tr>
<tr>
<td></td>
<td>Rpt Until</td>
<td>Usable</td>
<td>Repeat (action) Until (condition)</td>
</tr>
<tr>
<td></td>
<td>While</td>
<td>Usable</td>
<td>While (condition) Do (action)</td>
</tr>
<tr>
<td>Graphics</td>
<td>–</td>
<td>Usable</td>
<td>Graphics displayed on screen</td>
</tr>
<tr>
<td>Help</td>
<td>–</td>
<td>Usable</td>
<td>Provides help on item selected</td>
</tr>
<tr>
<td>HotSpot</td>
<td>–</td>
<td>Supportive</td>
<td>Hot spots defined on the still</td>
</tr>
</tbody>
</table>
### CHAPTER 4. HYTIME DESCRIPTION OF INTEGRATOR

<table>
<thead>
<tr>
<th>Input</th>
<th>Key</th>
<th>Mouse</th>
<th>Link</th>
<th>Options</th>
<th>PresentObject</th>
<th>PreviousLevel</th>
<th>Pool</th>
<th>By Name</th>
<th>By Type</th>
<th>Audio</th>
<th>Composite</th>
<th>Computation</th>
<th>Still</th>
<th>Video</th>
<th>Selector</th>
<th>HotSpot</th>
<th>Key</th>
<th>Menu</th>
<th>Mouse</th>
<th>StillImage</th>
<th>Synchro</th>
<th>Text</th>
<th>Tie</th>
<th>Transition</th>
<th>Audio</th>
<th>Still</th>
<th>Video</th>
<th>Window</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

|       |     |       |      |         |             |              |      |         |         |       |           |            |       |       |          |         |     |      |        |           |        |      |     |          |       |       |       |        |

#### 4.1.2 Hierarchy of Multimedia Application Objects

Following is the hierarchy of multimedia applications objects based on the functions they are used for and the methods that can be applied to them. Figure 3.4 shows the different classes under which the multimedia objects are organized. The first class, "Basic", contains...
CHAPTER 4 HYTIME DESCRIPTION OF INTEGRATOR

Input objects, Output objects, and Composite objects. The set of objects in this class is used to input messages to the application, output results during presentations, or perform both actions. The second class, “Control”, is used to affect the flow of control and data within applications. The third class, “Logical”, is used to support developers building an IMMA. These objects, with the exception of the Help object, are supportive by nature and cannot be included in the IMMA. The final class, “Relation”, is used to link and synchronize multimedia application objects during the presentation of applications.

Component>
  Basic>
    Basic Input>
    1. Computation
       Input>
    2. Clock
    3. Key
    4. Mouse
    Selector>
    5. HotSpot
    6. Key
    7. Menu
    8. Mouse

Basic Output>
  9. Animation
  10. Audio
  11. Graphics
  12. Stills
  13. Text
  Transition>
  14. Audio
  15. Stills
  16. Video
  17. Video
  18. Windows

27
CHAPTER 4. HYTIME DESCRIPTION OF INTEGRATOR

Composite>
  19. Composite Input
  20. Composite Interactive
  21. Composite Output

Control>
  Baton>
  22. Master (Main)
  Slave>
  23. Score
  24. Track

Conditional>
  25. Do While
  26. For Loop
  27. Goto
  28. If Then Else
  29. Loop N
  30. On X Do
  31. Repeat Until
  32. While

Logical>
  33. Tie

Help

Options>
  Execute
  Present Object
  Previous Level

Pool>
  By Name
  By Type>
  Audio
  Composite
  Computation
  StillImage
  Video

28
CHAPTER 4. HYTIME DESCRIPTION OF INTEGRATOR

Relation>
35. Hyper (Link)
36. Synchro

The hierarchy that is listed above follows an alphabetic listing of objects within a particular class/subclass. If viewed as a tree, the hierarchy represents all "usable" multimedia objects as the leaves of the tree (these have been numbered), and all the "supportive" objects at higher levels. The only exceptions to this are the leaves of the nodes "Options" and "Pool" that are considered as "supportive" instead of "usable". The HyTime listing of the objects in the Appendix follows the same order as shown above.

4.2 Synchronization and Timing Aspects in the Integrator

SYNCHRONIZATION

This section describes synchronization issues and problems, lists examples that illustrate important synchronization situations, and proposes solutions to solve these problems.

4.2.1 Temporal Arrangements of a Pair of Events

There are thirteen different ways in which a pair of events, for example, a still image (A) and an audio sequence (B), can be arranged [4]. These are shown in Figure 3.5. The HyTime listings of these situations have been described in detail in the Appendix of [52].

4.2.2 Some Rules for Placement of Multimedia Objects in Time and Space

When developing an IMMA, some rules for placement of multimedia objects in time and space must be followed for a consistent and correct description of the application. These provide a framework for a consistent design when the developer builds an IMMA. These rules were formed by considering examples from the class of applications that has a linear
CHAPTER 4. HYTIME DESCRIPTION OF INTEGRATOR

Figure 4.1: Hierarchy of Multimedia Application Objects
CHAPTER 4. HYTIME DESCRIPTION OF INTEGRATOR

Figure 4.2: Temporal Ordering of Two Events
document structure. The rules are formed such that these are sufficient for describing moderately complex IMMAs. These rules are:

(i) Placing Similar Objects Rule

Text objects, still images, animation, graphics and video are considered as similar in nature (reason why these can be placed on the same track in a score).

(ii) Overriding Attribute Values Rule

The values of an object that is defined and executed at a later time override the values of a similar object that has been executed prior to it.

For example, as shown in Figure 3.6a, if for a screen size of 640 x 480, a window 512 x 460 has been defined, where different still images of 512 x 460 are displayed sequentially, the second image displayed will override the first image thus overlapping and erasing the first image. The third image will erase the second, and so on, till the last image is displayed on the screen. In a variation of the above case, when the image size is the same as screen size, the values of the attributes for the second image would override the first image, the values of the third image would override the second and so on. When the window sizes are different such that they only partially (see Figure 3.6b) or don't overlap (see Figure 3.6c) at all, then the previous image will also be shown on the screen.

(iii) Handling Discrete Events Rule

Dialog boxes and help screens are considered as discrete events and can overlap (partially or fully) other objects. But these do not erase or override permanently the values for other objects as these are considered as interrupts to the applications and will be removed after their use.

(iv) Overlapping Audio Object Rule

Since objects can be defined (in the Integrator) on different tracks to be executed in parallel, one could have a situation where an audio object can overlap (in the time axis)
CHAPTER 4. HYTIME DESCRIPTION OF INTEGRATOR

any object including video (with no inherent audio tracks) as shown in Figure 3.6d, but would override the values if overlapped with video with inherent audio tracks. As shown in Figure 3.6e, the video object starts displaying frames on the screen and plays its own audio for some time. But when the audio “Audio1” starts playing, this object overrides values for the previous audio and starts to play even though the frames of the video are still displayed. When Audio1 stops, the audio within the video starts to play again. But in the other situations, where audio overlaps objects such as still images, graphics, etc., there is no problem during overlapping, as these are executed on different logical input/output tracks within the Integrator.

(v) Placing Transitions/Modifiers Rule

Transitions are modifiers and affect only those objects to which they are attached. These cannot exist on their own.

(vi) Handling Object Granularity Rule

Multimedia objects can be synchronized at various levels of granularity. The current version of the Integrator supports object level synchronization for all objects. This means that usable objects such as still images, graphics, etc., can be synchronized to either the start or the end of other usable objects.

4.2.3 Types of Synchronization in the Integrator

The Integrator allows flexibility to define different types of synchronization amongst objects when the developer defines an IMMA. The different types of synchronization that can be defined using the Integrator are as follows:

**Elementary Synchronization** is of two types (i) Serial/Sequential synchronization, where the different objects are sequentially placed on the track and execution of these objects takes place in a sequential manner. Cases 1, 2, 3, and 4 in Figure 3.5 are examples of this type. (ii) Parallel Synchronization, where objects are placed on different tracks and
CHAPTER 4. HYTIME DESCRIPTION OF INTEGRATOR

Window 2
(a) Fully Overlapped windows

Window 2
(b) Partially Overlapped windows

Window 2
(c) Non Overlapped windows

Audio 1
Video with no Audio
(d) Audio overlaps Video (no inherent audio tracks)

Audio 1
Video with Audio
(e) Audio overlaps Video (inherent audio tracks)

Display Screen

Figure 4.3: Illustrations Regarding Some Rules for Placement of Multimedia Objects in Time and Space
CHAPTER 4. HYTIME DESCRIPTION OF INTEGRATOR

tied together either explicitly, using the “Tie” object, or implicitly, by defining the time attribute within the object.

Pre-Synchronization where objects are synchronized in such a manner that their start times are the same. These objects will start at the same time, but not necessarily end at the same time. Cases 9 and 10 in Figure 3.5 are examples of this type of synchronization.

In-Synchronization where objects have the same starting and ending times. Case 13 in Figure 3.5 is an example of this type. The objects start at the same time and are executed for the same duration, thus having the same end times.

Post-Synchronization where objects are synchronized in such a manner that they end at the same time although they could have started execution at different times. Cases 11 and 12 in Figure 3.5 are examples of this type of synchronization.

Independent/No Synchronization where objects in the application are not tied or synchronized in any manner. For example, in an example such as that shown in Figure 3.7a, where a HyperCard template has buttons that are linked to different multimedia objects, objects are executed in the manner determined by the end-user’s selection of buttons. Thus there is no synchronization between text, video, audio, or still image object.

Conditional Synchronization where objects are synchronized in a manner depending on some condition. For example, using the “If-then-else” construct we can specify a Conditional synchronization such as:

```plaintext
if (condition)
  then (A pre-synchro B)
  else (A post-synchro B)
endif
```

The condition can be a simple comparison such as greater than, lesser than, equal to, or could use Boolean operators such as AND, OR, NOT, to make up a complex conditional statement.

Cyclic Synchronization (based on the type explained in [73]) applies to only one multimedia object which gets presented repetitively depending on the iteration value. For
CHAPTER 4. HYTIME DESCRIPTION OF INTEGRATOR

This is an example of a situation where there is no synchronization between the video segment, audio segment, still image, and the text to be displayed. The execution of the particular object depends on the end-user's action on selecting and triggering the button.

(a) Independent (No) Synchronization

(b) One-way Interruptible Synchronization

(c) Mutual Interruptible Synchronization

Figure 4.4: Independent and Interruptible Synchronization of Objects
CHAPTER 4. HYTIME DESCRIPTION OF INTEGRATOR

example,

```
loop 10
  play audio1       - play this audio 10 times -
endloop
```

The mapping of a similar construct shown in the last section of this chapter, deals with the mapping of Integrator notation to HyTime notation.

**Chained Synchronization** (based on the type explained in [73]) applies to more than one multimedia object (Basic output or Basic Input), the presentation of which will be chained together in some manner. A solution is to extend the application of elementary synchronization, introducing recursivity (through the use of the loop construct) in the definition of the synchronization. For example,

```
loop 20
  A pre-synch B       - A and B start at the same time -
  B post-synch C     - B and C end at the same time -
endloop
```

The mapping of a similar construct shown in the last section of this chapter, deals with the mapping of Integrator notation to HyTime notation.

**Interleaved Synchronization** where the extent of the objects being synchronized can overlap to some extent, but not fully. Cases 6 through 12 in Figure 3.5 are examples of this type of synchronization.

**Interruptible Synchronization** (based on the type explained in [60]). An interruption occurs when the activity of an action is suspended before the end of its lifetime and the activity of another action is begun in its place. Task interruption usually occurs due to actions initiated by the end-user, but they can also be the result of system initiated actions such as clock update, announcement of an e-mail message, etc. [60].

Figure 3.7b shows an example of a single-way interruptible synchronization, where the help object is called by the end-user. This interrupts the video "Video1", and temporarily
CHAPTER 4. HYTIME DESCRIPTION OF INTEGRATOR

halts display of the video. When the help object is closed the video sequence resumes. The requirement in this case is that the interrupting action must be completed for the interrupted action to restart.

Figure 3.7c shows an example of a two-way interruptible synchronization, where for example, 90 frames of audio keep interrupting the video sequence “Video1” and 200 frames of the video sequence interrupt the audio sequence “Audio1”. In this case both objects are temporarily halted at specific intervals of their execution. In the case of the Integrator, whenever an object is interrupted, the Virtual Time Unit (VTU) count associated with it is also interrupted till the interrupted object is restarted. Thus, the mapping of VTU to Real Time Unit (RTU) is halted as long as the object is in the interrupted state. This allows the developer to correctly map the time associated with the particular object.

4.2.4 Other Issues Regarding Synchronization

Other issues are illustrated through five illustrative examples.

(i) Example: Synchronize objects on a single track in a score. This involves serial/sequential scheduling of events (as shown in Figure 3.8a) and can be handled by representing a core event sequence (sequence of events to be executed) that contains objects that are to be scheduled. The baton (function to convert Virtual Time Unit to Real Time Unit) representing the track, on which these objects have been placed, counts time in VTUs. As VTU values are known, these can be mapped to RTUs as time progresses. The rules (placement of multimedia objects) (i), (ii), and (v) must be followed in such cases of synchronization.

(ii) Example: Display the still image after 70 seconds of audio has been played. This situation involves synchronizing two objects (still image and an audio sequence) on different tracks in a score (see Figure 3.8b). This case could be solved by either having a single baton controlling both tracks or by having two different batons controlling the two tracks on which objects have been placed. In the first case, the granule size of both the VTUs would be the same and would map to the Target Finite Coordinate System (FCS)
CHAPTER 4. HYTIME DESCRIPTION OF INTEGRATOR

(a) Serial / Sequential scheduling of events

(b) Simple parallel synchronization of events

(c) Synchronization of multiple events

Figure 4.5: Simple Situations Concerning Synchronization
CHAPTER 4. HYTIME DESCRIPTION OF INTEGRATOR

with the same RTU granule size (in this case, seconds). So, when the audio starts to play, the application checks the system clock every second and displays the still image when 70 seconds are up. The rules for placement of multimedia objects must be followed, only in this case it does not matter since the audio causes no problems when synchronized with the still image, as they are in different logical output tracks. In the second case, the VTU values of both batons get mapped to RTU values separately. Assume that in this case, the granule size of the RTU of the Audio is in "minutes", and the granule size of the still image is in "seconds". The system then converts the granule size of all active batons to the lowest granule size, in this case, "seconds". The application checks the system clock every second (instead of minute), and then displays the image at the appropriate time.

Note 1: The system must convert the granule size of the RTU to the smallest size used in the application to correctly effect timing and synchronization of the objects.

(iii) Example: Three slave batons (baton1, baton2, and baton3) control three tracks in the score (as shown in Figure 3.8c). The granule size for RTU for baton1 is in seconds, and for baton2 and baton3 are in minutes. Audio1 and Animation1 overlap (Animation1 during Audio1), and Audio2 and Video1 partially overlap. Audio1 and Audio2 are controlled by different tempos with baton2.

The system first changes the granule size of all active batons to seconds. There is no problem of synchronizing Animation1 and Audio1 due to the rules for placement of multimedia objects, but the partial overlap of Audio2 and Video1 causes the last part of the audio inherent in Video1 to be cut off due to the early start of the Audio2 audio sequence. To avoid this we can make use of a tie object that ties the end of the Video1 object to the start of the Audio2 object, thus making sure that the audio sequence starts only after the video sequence is over. Tieing the two objects has another advantage. Suppose the end-user interrupts the system by, say, calling the help object, then Audio2 would not be rigidly bound to the time line but is instead dependent on the video sequence due to the tie.

(iv) Example: This consists of a menu display with four choices: a slideshow, presen-
CHAPTER 4. HYTIME DESCRIPTION OF INTEGRATOR

Figure 4.6: Complex Situation Concerning Synchronization

tation of a video, presentation of an audio segment, and quit (see Figure 3.9). The menu is placed within a loop object that makes the application to return back to the menu after the user selects some choice other than the quit option. If the user selects quit, the application stops. If the user selects any of the other three choices, the application executes that selection and then returns back to the menu display. The figure shows the different batons (baton1-baton5) that govern different tracks (window1,2,4-plus audio and control).

Here, all the rules for placement of multimedia objects have to be followed and the system converts the RTU granules of all objects to the smallest granule, i.e., “seconds”. When the audio menu item is selected, Audio start to play, but the menu is still on the screen (due to rule for placement of multimedia objects (iv)). If now the user selects the video menu item, the Video also starts, erasing the menu off the screen and overriding the
CHAPTER 4. HYTIME DESCRIPTION OF INTEGRATOR

Audio values, thus stopping the audio play. The VTU values for Audio gets reset to the original values. When the Video stops, VTU values for the Video are again reset to the original values, and the menu is once more displayed (due to the loop condition). The system clock keeps running, but the mapping of VTU to RTU stops when the objects finish executing.

Suppose at a different time, when the audio menu item is selected, the Audio starts playing, and the menu is still displayed on the screen. Now, if the user selects the slideshow menu item (before the Audio ends), the slideshow starts, erasing the menu, but there is no synchronizing problem with the Audio (due to rule (iv)). The slideshow lasts longer than the Audio (whose VTU values get reset once it stops playing), and when the last still image is displayed for the time set in its time attribute, the system resets the VTU values of the slideshow and then displays the menu (which now overrides the values set by the last still image).

Note 2: The system resets VTU values of the objects to their original values when Loop, Conditional objects, etc., are used so that the application is ready for execution for the next iteration/condition.

(v) Example: Show how Interrupts (Pauses) affect the synchronization of application objects.
Suppose in the example shown in Figure 3.9, if the user clicks the right mouse button, all processes stop and the application displays a message: “Application Interrupted - Click left mouse button to begin”. When the user clicks the left mouse button all interrupted processes start executing from the interrupted state.

When the user starts both the audio sequence and the slideshow applications, and later clicks the right mouse button, the system stops the VTU to RTU mapping for all the active batons (i.e., baton1, baton3, baton4, and baton5) and displays the message. Now all active processes are in an interrupted state and wait for the user to restart by clicking the left
mouse button. When the user does so, the mapping process begins again and the application resumes execution.

**Note 3:** The system only temporarily halts the mapping of the VTU to RTU when application interrupts occur and does not reset VTU to original values. When the interrupting condition is over, the system restarts the mapping process thus preserving correct time values for objects in the application.

**TIMING**

The rest of section 3.5 describes Timing issues and problems related to the Integrator, lists examples that capture important timing situations, and proposes solutions to solve these problems.

### 4.2.5 Conversion of VTU to RTU

Time values for all Integrator objects are defined in VTU. This has to be mapped to RTU during execution of the application. The mapping from VTU to RTU is done by event projectors each called “Baton”. The baton is a list of tempo directives that define the number of RTUs that pass for a given VTU at any particular point in virtual time [97]. There are two cases regarding the tempo definition in the tempo directives: (i) tempo is constant throughout the directive or, (ii) tempo varies over time.

When the tempo is a constant, we can simply multiply the virtual time duration of an event by the rate of change of time given at the start of the tempo. This would yield the RTU value for the event, i.e.,

\[ \text{vtu} \times \frac{\text{rtu}}{\text{vtu}} = x \cdot \text{rtu} \]

where \( x = \text{number of RTUs that pass for 1VTU at a given virtual time.} \)

When the tempo changes over time, the value for the RTU is determined by performing the definite integral of the function that defines the rate of change. A detailed discussion is given in [97].

43
CHAPTER 4. HYTIME DESCRIPTION OF INTEGRATOR

Figure 4.7: Using Batons to Project Time from One FCS to Another
CHAPTER 4. HYTIME DESCRIPTION OF INTEGRATOR

A pictorial representation of this translation (mapping) from VTU to RTU is shown in Figure 3.10a. The audio event (which has been defined with 10 VTUs), in the Source FCS, is projected to the Target FCS using the baton with the tempo defined as 4 RTUs per VTU. This in effect projects the audio sequence to be played over 40 RTUs.

4.2.6 Controlling of Batons

Figure 3.13 shows an application made up of several scores. The topmost score (main score) contains the master baton which controls the timing for all tracks in the main score and in turn tracks in the lower level (slave) scores. This is possible because the RTU of one baton can be passed through as the VTU to the next baton, thus controlling execution of the second baton. This is shown in Figure 3.10b. Each baton can be thought of as a function that converts a duration in virtual time to a duration in real time. A detailed explanation is given in [96].

In the case of the Integrator, one could leave the timing open (not define any time value for the object) for some composite object, for example, like that in Score 2 in Figure 3.13. This would cause the other objects (that may be defined) in the (lower level) Score 4 to be executed first. Due to this, the VTU of the objects in Score 4 is mapped to RTUs which in turn are passed on as the VTU to the baton in Score 2. This then gets mapped to RTUs and is passed to the main score where the TimeLine controls the execution of the object. If we define the time for a composite object that is in a higher level score, for example, like that shown in Figure 3.11a, it will affect execution of the lower level scores in the manner described below. Here, the time taken to execute the composite is set to 15 minutes, which in turn, controls the still images to be displayed in 15 minutes. The pictorial description of the mapping from Source FCS to Target FCS shows that, when still images are mapped finally, the total RTU value would be 15 minutes, irrespective of the mapping function defined in the lower level baton.
CHAPTER 4. HYTIME DESCRIPTION OF INTEGRATOR

Figure 4.8: Defining Time Value for Composite Objects
4.2.7 Other Issues and Solutions

(i) Pause and Restart operations.
Whenever an interrupt condition occurs in an application, all timing processes (except the real-time system clock) are interrupted. The mapping mechanism for all active batons are in a halted state and are resumed only when the interrupting condition is over. Example (v) in the Synchronization subsection (section 3.5) shows an application where interrupting conditions are handled. The Restart operation is done when the interrupting condition is no longer present. This way all (temporarily) halted processes can begin their execution from the interrupted state.

(ii) Reset Operation.
This operation is necessary in those situations where the application uses looping or conditional mechanisms to control objects. When a particular iteration (for example, in a loop) is completed, tempo values for that section of the baton are reset (using the Reset operation). This is done so that, when the next iteration is executed, the same values for the tempo would be used. It does not matter whether the tempo has a constant value or whether it varies over time, since the same (previous) value would be used for subsequent iterations.

(iii) Menu selection.
When a menu selection is performed, tempos for the baton governing the track containing the action to be performed are executed. All other tempos remain in the same state as they were that particular time. When the slideshow (see Figure 3.9) is selected from the different menu choices present in the menu display, only tempos in baton 1 are used for execution. Tempo directives in the other batons remain in the same state before menu selection was done.

4.3 Mapping Representations: Integrator to HyTime

This section describes mechanisms for mapping the Integrator notation to HyTime notation. We list various constructs and objects used within the Integrator, define their functions,
CHAPTER 4. HYTIME DESCRIPTION OF INTEGRATOR

and show by way of examples how the mapping is performed. The HyTime notation is used as an intermediate representation for defining and describing IMMA s. The intermediate representation can be parsed and stored in some system such as LEND [38]. Links, script expressions, functions, etc., within the document describing an IMMA are resolved by executing a HyTime engine that accesses the database (within LEND) for information on contents of the document. The representation builder takes run-time information as the Integrator runs and maps that into LEND structures. The unparsers or generator exports HyTime documents that it builds from the information in LEND. Each such executable document consists of the HyTime part and the script/expression part as shown in Figure 3.12. The script part gets information about synchronization, scheduling, etc., from the HyTime part when the document is executed to present the application.

4.3.1 Mapping Composite Object to HyTime Notation

Composite objects in the Integrator are objects that can contain from just a single object to an entire score. These objects give the flexibility to the developer to transfer control from one part of a score to another. A score in a composite is considered in a lower level when compared to the level of the score containing that particular composite. This gives a tree structure with the main score at the root of the tree as shown in the Figure 3.13. Flow of execution will be from left to right in a pre-order traversal of the tree structure consisting of the scores.

A composite object gets mapped to a "special core event". The element declaration for a composite is (see Appendix for a full HyTime listing):

```
<! ELEMENT composite - - (sce) >
<! ATTLIST composite
 - -include composite attributes-
 - -see objects listing in appendix-
>
<! ELEMENT sce - - (compin | compintr | compout)+ >
<! ATTLIST sce
 - -include HyTime attributes-
```
CHAPTER 4. HYTIME DESCRIPTION OF INTEGRATOR

Figure 4.9: Integrator Architecture
CHAPTER 4. HYTIME DESCRIPTION OF INTEGRATOR

Figure 4.10: Flow of Execution through Scores Containing Composite Objects
Elements such as ce - "core event", evseq - "event sequence", and ces - "core event sequence" are HyTime elements that are defined in the HyTime meta-DTD. The composite object can be of three types: composite input, composite interactive, and composite output (represented as "compin", "compintr", and "compout" as shown in the ELEMENT declaration above). The difference between these objects are in the types of core events that can be allowed within the composite object. The composite input object allows only "input" objects to be used within the composite, the composite output object allows only "output" objects to be used within the composite, whereas the composite interactive object can use both "input" and "output" type objects within the composite. The above "ELEMENT" declaration states that a composite object is a "sce", which in turn must be one or more "compin", or "compintr", or "compout".

4.3.2 Mapping Computation Object to HyTime Notation

The computation object is used in the IMMA to allow externally compiled programs to perform some computation on objects used within the IMMA, and to send results back to the application. The external program may be a simple function, such as some mathematical formula, or it could be a complex program, for example, a program to decompress and display images in some special format.

The computation object is mapped to the HyTime notation through the "NOTATION" and "ENTITY" declarations such as those listed below. For example, to use an external program to display images in the DVI format, we could declare:

```xml
<!ENTITY Picture1 SYSTEM "/usr/arun/slides/slidel" NDATA DVI >
<!NOTATION DVI SYSTEM "/usr/bin/vshow.exe" >
```

Here Picture1 is a still image in the DVI format. The vshow.exe program is called
CHAPTER 4. HYTIME DESCRIPTION OF INTEGRATOR

through use of a Computation object to display the image. The document instance that invokes this call and displays the image would then look like:

```xml
<integevent exspec="dcompu">
  <see>
    <compout>
      <visevent id="dcompu" exspec="dstill"> <visobj src="Picture1"/> </visobj> </visevent>
    </compout>
  </see>
</integevent>
```

The above set of statements specify that the integrator event “integevent” consists of an “see” element, which in turn consists a composite output object “compout”. The compout object calls the visual-event element “visevent” to display the still image (whose source is “Picture1”).

4.3.3 Mapping Conditional Object to HyTime Notation

Different conditional objects within the Integrator are used to evaluate conditions under which certain actions are performed and other actions ignored. The different types of conditional objects are DoWhile, ForLoop, Goto, IfThenElse, LoopN, OnXDo, RepeatUntil, and While.

To map the conditional objects to HyTime we have two alternatives: (1) declaring tags for each component of the conditional object and including the contents within these sets of tags, and, (2) declaring a notation for these objects and having a controlling program to execute the contents of the objects. The latter alternative is more elegant than the former and gives the developer a general document structure, since, by changing the notation declaration the controlling program can be changed. This leaves the developer to make only minor changes to the Document Type Definition (DTD), as shown in the example below. We describe the mapping for both alternatives, and leave it to the discretion of the developer to choose whichever is suitable to the application being built.

(1) The DTD declaration for a DoWhile conditional object, when tags are defined for
CHAPTER 4. HYTIME DESCRIPTION OF INTEGRATOR

the components, looks like:

<! ELEMENT dowhile - - (action? | cond) >
<! ATTLiST dowhile
   - -include dowhile attributes-
   - -see objects listing in appendix-
>
<! ELEMENT action - - (#PCDATA) - -action statements- ->
<! ELEMENT cond - - (#PCDATA) - -condition statements- ->

The document instance containing the DoWhile element would be:

<dowhile>
   <action>... </action>
   <cond>... </cond>
</dowhile>

(2) The DTD declaration for the DoWhile object when a notation is declared looks like:

<! ELEMENT script - - (#PCDATA) >
<! ATTLiST script
   - - Integrator attributes - -
   Integrator attributes
   Integ CDATA #FIXED "script" - -attributes specific to integrator-
   id ID #IMPLIED - -id of script object-
   file IDREF #CONREF - -file referred to by the script object-
   notation NAME #REQUIRED - -data content notation included in the script object-
>

The document instance containing the DoWhile conditional element and controlled by, for example, a MEDIAscript program, would be:

<script id=abc notation=mediascript>
   do {
      cls black /*clear screen to black*/
      image slide1 /*load image in buffer*/
   }
</script>
CHAPTER 4. HYTIME DESCRIPTION OF INTEGRATOR

```c
wipe horiz /* wipe image onto screen */
aud audio1 /* start audio */
wait 900 /* wait 30 seconds */
cls black /* clear screen to black */
} while (spacebar)
</script>

while one controlled by, for example, a program written in C, would be:

```c
<script id=abc notation=ccode>
do {
    cls(black); /* clear screen to black */
    vshow(slidel); /* display image using vshow routine */
    vaudplay(audio1); /* play audio using vaudplay routine */
    wait(30); /* delay for 30 seconds */
    cls(black); /* clear screen to black */
} while (getkey = space)
</script>
```

This gives the developer flexibility to reuse the same DTD (declared for the document using this object) for all other conditional objects as well. In case 2 above, when notations are used, the number of elements that are declared for a document is less, and any available external program can be used to control the document.

4.3.4 Mapping Link (Hyper) Object to HyTime Notation

This object is used to link one object to another in an IMMA. This is mainly used when the user gets to navigate in a document environment, for example, jumping from one location to another in order to get more information on some item or moving around a document by activating and triggering buttons.

This object can be directly mapped to HyTime using the rich set of constructs available in the Hyperlinks module in HyTime. HyTime also gives a number of alternatives to represent linking of objects. Some of the different types of link that are available are ilink,
clink, agglink, and plink. They can be used in different ways depending on the application. The following example shows how an audio segment can be linked to some text object. This example is a subset of the example listed in [111].

The document instance

```xml
<txtobj id="jack">
  <audlink linkend="nsaw">house that jack built</audlink>
</txtobj>
```

is an example of a clink where the audio sequence “nsaw” is linked with the text object “house that jack built”. The audio sequence is played when the text is displayed on the screen. The DTD would have to include the following statements along with the rest of the declaration in order to support this example:

```xml
<!ENTITY % link "subend | audlink" >
<! ELEMENT audlink -O ANY > -clink definition-
<! ATTLIST audlink
  HyTime NAME #FIXED clink
  id ID #IMPLIED -id of object-
  refsub CDATA #FIXED "subend"
  linkend IDREF #REQUIRED >
<! ELEMENT subend -O (#PCDATA) > -link end definition-
<! ATTLIST sub-end
  HyTime NAME #FIXED linkend -include HyTime linkend attributes-
>
```

### 4.3.5 Mapping Loop/Iteration Object to HyTime Notation

The loop/iteration construct in the Integrator allows for repetitions of actions and gives a looping mechanism to execute actions some number of times, depending on the condition. The different types of Loop constructs are:
CHAPTER 4. HYTIME DESCRIPTION OF INTEGRATOR

- loop a fixed number of times - LoopN,

- loop number of times depending on the value of the variable - LoopVar, and

- loop number of times depending on the condition - LoopCond.

The loop/iteration construct is mapped into HyTime in a similar manner to the conditional object. One could define notations for this object and declare controlling programs to execute this object or declare a set of tags for the contents of the loop/iteration construct. However, to declare a set of tags for the loop construct, we would have to declare the following:

```xml
<!ELEMENT loop (cond & statement+)>
<!ATTLIST loop
   include loop attributes -
   see objects listing in appendix -
>
<!ELEMENT statement (#PCDATA)>
<!ATTLIST statement
   include statement attributes -
   see objects listing in appendix -
>
<!ELEMENT cond (#PCDATA)>
<!ATTLIST cond
   include cond attributes -
   see objects listing in appendix -
   iterate NUMBER #CONREF>
```

An instance of a document containing the loop construct would look like:

```xml
<loop>
   <cond iterate=20> <- N=20 ->
   <statement> ... </statement> <- loop this statement 20 times ->
</loop>
```

The “&” condition in the ELEMENT declaration for the loop is necessary to have just one construct for both top-of-loop test and bottom-of-loop test.
CHAPTER 4. HYTIME DESCRIPTION OF INTEGRATOR

4.3.6 Mapping Usable Objects and their Attributes to HyTime Notation

Those objects that are categorized as “usable” objects (please refer to earlier section on Objects in this chapter) get incorporated directly in the IMMA. These objects form the core of the final application presentation where the end-user gets to interact with them as the application is run.

These objects get mapped to “objects” in the HyTime notation. By incorporating time and position attributes to these objects, they are transformed into “events”. Integrator objects that are examples of this category are Animation, Audio, AudioTransition, Clock, Computation, Graphics, Help, Key, Mouse, StillImage, StillTransition, Text, Variable, Video, and VideoTransition. The HyTime declarations for some of these objects are shown below. All other usable objects get mapped in the same manner.

Animation:
<!ELEMENT animate - O EMPTY >
<!ATTLIST animate
  animate
    - -include animate attributes-
    - -see objects listing in appendix-

>

Clock:
<!ELEMENT clock - O EMPTY >
<!ATTLIST clock
clock
  - -include clock attributes-
  - -see objects listing in appendix-

>

Text:
<!ELEMENT text - O EMPTY >
<!ATTLIST text
  text
    - -include text attributes-
    - -see objects listing in appendix-

>

Window:
<!ELEMENT window - O EMPTY >
<!ATTLIST window
  window
    - -include window attributes-
    - -see objects listing in appendix-

57
For a full listing of these objects and attributes please refer to the Appendix. Those objects that are categorized as supportive objects do not get mapped to the HyTime notation since they cannot be used in the IMMA and hence in the document that represents the IMMA. This includes Basic, Basic-Input, Basic-Output, Components, Conditional, Control, HotSpot, Input, Logical, Option, Option-Execute, Option-PresentObject, Option-PreviousLevel, Pool, Pool-ByName, Pool-ByType, Relational, Selector, Selector-HotSpot, Selector-Key, Selector-Menu, Selector-Mouse, Slave, Slave-Score, Slave-Track, and Transition. The only exception to this is the Window object which is considered “usable” and can be mapped into the HyTime notation.

4.3.7 Mapping Orelation (Run-time step) Attribute to HyTime Notation

The orelation attribute declared in the Integrator objects keeps track of (possible) events that would get executed subsequently. In the Integrator this relationship can be found by analyzing time attributes set for the various objects. In some cases it may not be possible to know which object would get executed next, as in the case where the end-user is given the choice of selecting something from a menu.

As there is no direct mapping for such a construct, the developer would have to encode it using the different options in HyTime. The modules that are used to map this construct are the Finite Coordinate System (FCS) module, Location Address module, and the Hyperlinks module. The elements from within these modules that help in the mapping include namelec, bibloc, agglloc, treeloc, pathloc, tokenloc, illinks, clinks, fcs, axis, and evsched. The choice of one or more of these elements strictly depends on the application condition as there are several ways to encode a single construct. For example, when all the names/ids of the objects that are used in an application are known, one can use the namelec element to step through the different events to find the order of execution. For the same case, someone else might prefer to use the treeloc element to step through the events. To step through
CHAPTER 4. HYTIME DESCRIPTION OF INTEGRATOR

a sequence of events, one could also use a "core event sequence" when the objects to be executed are placed in a sequential manner. The FCS module helps in keeping track of the axis along which the events have been ordered, and the event schedules in which they have been declared. The example to review documents listed in [111] gives a flavor of how events can be accessed in more than one way. A subset of that example is listed below:

The DTD for such a class of documents includes declaration for the following along with other declarations that support the rest of the document.

```
<! ENTITY % loc "nameloc | tokensloc | treeloc" >
<! ELEMENT dimspec - O (marker1?, marker2) >
<! ELEMENT tokenloc - O (dimspec+) >
<! ELEMENT marker1 0 0 (marker) >
<! ELEMENT marker2 0 0 (marker) >
<! ELEMENT marker 0 0 (marker) >
<! ATTLIST (dimspec, tokenloc, marker1, marker2, marker)
- -include (dimspec, tokenloc, marker1, marker2, and marker) attributes- -
- -see objects listing in appendix- -
```

The document instance contains:

```
<treeloc id="trefl" locsrc="grapesid">
  <dimspec><marker1><marker> 2 <marker2><marker> 1 </dimspec>
  <dimspec><marker1><marker> 4 <marker2><marker> 1 </dimspec>
  <dimspec><marker1><marker> 3 <marker2><marker> 1 </dimspec>
</treeloc>
<treeloc id="eureka" locsrc="trefl">
  <dimspec><marker1><marker> -2 <marker2><marker> 2 </dimspec>
</treeloc>
```

where the treeloc element locates an unnamed subelement by its position relative to the root element. The dimension markers, "marker1" and "marker2", are used to step through the elements in order to reach the required element in the document. Please refer to the HyTime standard [52], for a detailed explanation on the use of markers and marker
functions. The \texttt{tokenloc} element "eureka" locates the last two words (or tokens) in the subelement identified by the \texttt{treeloc} "tref1".

4.3.8 Mapping Score Construct to HyTime Notation

The score in the Integrator consists of a list of tracks.

The score gets mapped to a "core event sequence". A core event sequence contains a list of event sequences. An instance of a document containing such a core event sequence could be:

\begin{verbatim}
<ces>
  <evseq>
    <ce>
      <audio id="audio.seq"/>
    </ce>
  </evseq>
  <evseq>
    <ce>
      <audio id="audio1.seq"/>
    </ce>
    <ce>
      <audio id="audio2.seq"/>
    </ce>
  </evseq>
</ces>
\end{verbatim}

where the core event sequence in the above example contains a list of audio (event) sequences. There is no difference between the top level score and the lower level scores (if the IMMA is built using multiple scores) in the intermediate representation since the score has an "id" attribute that would identify the type of score in the application.

A DTD representing the definition for a score could look like:

\begin{verbatim}
<!ELEMENT  score - (ces)+  >
<!ATTLIST  score
- -include score attributes- -
\end{verbatim}
CHAPTER 4. HYTIME DESCRIPTION OF INTEGRATOR

- -see objects listing in appendix- -

Another way to map the score is to define it in terms of a schedule “sched”. In this case, each score maps to a schedule, which in turn contains the Integrator (multimedia) objects “integevent”, visual events “visevent” or audio events “audevent” (refer to Slide Show with User Control Example in the Appendix).

\[
\begin{align*}
\text{\langle sched\rangle} & \text{\langle! - evsched with basegran of 1/10 second - \rangle} \\
& \text{\langle visevent id=\"beginshow\" exspec=\"ds\" \rangle <visobj src=\"still1\">\text{\langle visevent\rangle}} \\
& \text{\langle audevent exspec=\"da\" \rangle <audobj src=\"audio2\">\text{\langle audevent\rangle}} \\
& \text{\langle integevent id=\"endshow\" exspec=\"dwait\"> <wait script=wait1>5</wait> \text{\langle integevent\rangle}} \\
& \text{\langle\text{/sched}\rangle}\end{align*}
\]

In the above example, the schedule “sched” contains three events “integevent”, “visevent”, and “audevent”. The objects that are defined within the events are a still image “stillim1”, audio object “audio2”, and a wait object.

4.3.9 Mapping Synchro Object to HyTime Notation

This construct is used in the Integrator to synchronize objects that may be on different tracks or in different scores. The different types of synchro are serial/sequential, parallel, pre/in/post synchro, independent, interruptible, conditional, cyclic, and chained.

The serial/sequential synchro is easily mapped into a “core event sequence”, where different core events are placed in a sequential way and gets executed in a sequential manner. The different temporal relationships discussed in the Timing and Synchronization section show how objects can be synchronized in parallel, in pre/in/post synchronized manner, independent and chained. Conditional synchronization is mapped using the “conditional” construct along with the objects that are to be synchronized. For example,

\[
\text{\langle ifthenel\rangle} \\
\text{\langle cond\rangle} \ldots \text{\langle/cond\rangle}
\]
4. CYCLESYNCH DESCRIPTION OF INTEGRATOR

A synch B
B synch A

Cyclic synchronization can be mapped in a similar way by using the "iteration/loop" construct to define the objects that are to be synchronized in such manner. For example,

\[
\text{<loop>}
\text{<cond iterate=20>}
\text{A synch B }<\text{/then>}
\text{B synch C }<\text{/else>}
\text{</loop>}
\]

\[\text{<! - N=20 ->} \text{- synchronize A B C in some manner for 20 times-} \]

4.3.10 Mapping Tie Object to HyTime Notation

This construct is used to tie two or more objects that are on different tracks. Objects that are tied together effectively slide together on separate tracks. This is mainly used to synchronize objects so that they start or end together.

The tie object cannot be mapped directly to the HyTime notation but can be encoded using several elements and constructs that are available in the measurement module in HyTime. Elements such as dimension specification, markers, marker function, and dimension reference can be used in combination to tie objects together and synchronize them in any manner. Also simple marker functions such as those shown in the Appendix in [52] would be very useful to compute functions and use them to tie objects. Some simple cases for tying two objects (for example, an audio and a video sequence) would be:

- video sequence starts when the audio sequence ends,
- audio sequence starts when the video sequence ends,
- both sequences begin at the same time,
- both sequences end at the same time,
CHAPTER 4. HYTIME DESCRIPTION OF INTEGRATOR

- both sequences start and end at the same time.

There are thirteen possible temporal relationships between two objects as shown in Figure 4.2 and in [4,52]. These have also been dealt with in the previous section on Timing and Synchronization. The declaration for a tie object would be:

```xml
<! ELEMENT tie (tstart | tend) >
<! ATTLIST tie
    - -include tie attributes- -
    - -see objects listing in appendix- -
>
<! ELEMENT (tstart | tend) -- (event, event, time)+ >
<! ATTLIST tiestart
    - -include (tstart and tend) attributes- -
>
```

where event defines the HyTime event used in the document and time defines the coordinate of the time axis declared for the application.

4.3.11 Mapping Top Level TimeLine and Time for Tracks to HyTime Notation

The top level time line keeps track of real-time clock pulses. The time unit is called a real time unit (RTU). The time for tracks keeps a count of the pulses in terms of virtual time. This time unit is called a virtual time unit (VTU). The VTU is mapped to RTU by an event projector called a “baton”. The time for a track in a higher level score acts as a controlling mechanism for those tracks that are in the lower level score as shown in Figure 4.10. The time units for tracks in the lower level score (considered as VTUs) are mapped into RTUs and then passed on as VTUs to the (controlling) track of a higher level score. This in turn gets mapped to RTUs and is passed on till the top level time line is reached.

The top level TimeLine gets mapped onto the “MasterBaton” in the HyTime notation, and the times for other tracks get mapped onto “SlaveBatons”. Although we differentiate
between the two types of batons in the Integrator notation, the HyTime declaration defines only a generic type called the “baton”, which contains an identifier attribute to differentiate between the master and the slave batons. The slave batons would also have a track identifier attribute, “trackid”, that identifies the track being controlled by a particular baton. Time specifications for all objects in the Integrator is in VTUs and these get projected to RTUs when the application is executed. The HyTime declaration for a baton is:

```
<! ELEMENT baton (proscope)* >
<! ATTLIST baton
   - - include baton attributes - -
   - -see objects listing in appendix- -
>
```

The document instance would contain, for example:

```
<baton id="slavebtn1">
  <tempo id="tempo1">
    <virdur><vtu>1</vtu></virdur>
    <realdur><rtu rtubase="60">60</rtu></realdur>
  </tempo>
</baton>
```

which declares a slave baton, “slavebtn1”, with a single tempo, whose 1 VTU maps to 60 RTUs. The rtubase declaration specifies that 60 RTUs map to 60 seconds (rtubase=60), thus mapping 1 VTU to 1 second in this example.

4.3.12 Mapping Tracks Construct to HyTime Notation

Tracks in the Integrator are logical input/output devices on which objects are placed to form the IMMA.

A track gets mapped to an “event sequence”. An event sequence can contain a list of core events. An instance of a document containing an event sequence could be:
CHAPTER 4. HYTIME DESCRIPTION OF INTEGRATOR

<evseq>
    <ce><audio id="audio.seq" > </ce>
    <ce><audio id="audio1.seq" > </ce>
    ...
    ...
    <ce><video id="video.seq" > </ce>
    <ce><video id="video1.seq" > </ce>
</evseq>

where the event sequence in the above example contains a list of audio and video core events.
Chapter 5

Conclusions and Future Work

5.1 Conclusions

This work involved the design of an authoring system called the “Integrator”, which helps developers to build interactive multimedia applications.

We have formalized the set of objects and the constructs used in the Integrator using the HyTime standard. We have explored the work done by the MHEG committee, defined a rich set of multimedia objects, and have shown how this set can be used with the HyTime standard to define multimedia applications. Some major timing and synchronization problems have been identified and have been listed along with proposed solutions to solve them. We have also shown a mapping mechanism, to map constructs used in the Integrator to HyTime notation, which we then use as an intermediate representation for describing IMMA applications. Some IMMA examples have been listed in both the Integrator and the HyTime notations to show the mapping between the two notations. These examples capture important situations regarding mapping, timing, synchronization, etc.

The motivation to use HyTime to represent IMMA was that HyTime is an application of SGML, which is a widely used ISO standard to define document structure. The advantage of using HyTime (which should become an international multimedia standard in April 1992) is that it addresses issues of standardizing multimedia applications and document exchange.
CHAPTER 5. CONCLUSIONS AND FUTURE WORK

between users of different platforms. This can be very helpful in cases where users may want to exchange IMMA documents but can't, since they might use platforms that may not support the Integrator. Also, the HyTime standard is extensive and defines rules (called architectural forms) to perform many common multimedia functions such as linking between objects, specifying time and mapping it to different axes, locating items within documents, modifying items before they are rendered, etc.

The process for mapping IMMAs from the Integrator to HyTime notation is currently done manually (which is quite cumbersome) and can be automated. HyTime notation allows applications to be broken down to smaller and simpler modules, and represents easily the contents of such modules. A major disadvantage with this approach was that even for a simple application, the intermediate representation could get quite large and be very complicated if the IMMA was complex.

5.2 Future Work

- Since this work focuses on designing the multimedia Integrator to work with HyTime, the next step would be evaluate and validate the formalisms described in this work. After this step one could implement this design on some system and then port it to other platforms. The Integrator architecture described has several components such as the representation builder, HyTime engine, user interface, unparsers, etc., that have yet to be fully built and tested.

- The existing prototype of the front-end of the Integrator does not do much to describe a complete IMMA. This work, and a related article [114], describes the components of the front-end which can be used as a starting point to build the interface to the Integrator.

- Although objects and their attributes used in the Integrator were designed to be general, they are by no means a complete set. This set can be further extended (either by adding other required objects or by extending the attributes for the current
CHAPTER 5. CONCLUSIONS AND FUTURE WORK

set of objects) to allow different platforms to support the Integrator. Also, a functional
description of the objects representing the relationship of objects with one another
can be described. Two classes of objects, (i) a pool that maps to a database in an
application, that can be searched and (ii) audio modifiers, “mixers”, that can be used
to control amplitude, select channels, etc., can be included in the future extension of
the objects set.

- Mapping tracks in the score to logical input/output devices and an efficient imple­
  mentation of handling multiple composites in an application also has to be defined.
- Timing and synchronization issues are by themselves a research problem and work
can be done in this area by taking some real life situations and examples like point
of sale applications, collaboration/co-operation between users of a computer system,
education/entertainment applications, etc.
- Tools required to automate the process of mapping the constructs from the Integrator
  notation to the HyTime notation can be built and extended later to support other
  platforms and applications.
REFERENCES


REFERENCES


REFERENCES


REFERENCES


REFERENCES


REFERENCES


[66] Hovey, R.B., Multimedia and Hypermedia Standardization, presented to the JTC1 TAG Committee, Washington D.C., October 12, 1990.


REFERENCES


REFERENCES


REFERENCES


REFERENCES


[125] Trigg, R.H., A Network-Based Approach to Text Handling for the Online Scientific Community, Maryland Artificial Intelligence Group, Department of Computer Science, University of Maryland, College Park, MD, 1983.


Appendix A

HyTime Listing of Objects

This appendix gives a formalized listing of the multimedia application objects that can be used by the developer to build IMMAs using the Integrator. This listing has to be included within the DTD specifying the IMMA document so that the developer can support these objects within the application.

```
<!- - Integrator Objects Definition - ->
<! ELEMENT compont - - (basic | control | logical | relation) >
<! ATTLIST compont - -
  Integrator Objects Definition - ->
<! ELEMENT Integrator Object - ->
<! ATTLIST Integrator Object - ->
```

```
<! ELEMENT Basic object - ->
<! ATTLIST Basic object - ->
<! ELEMENT Supportive Object - ->
<! ATTLIST Supportive Object - ->
```
APPENDIX A. HYTIME LISTING OF OBJECTS

<! ELEMENT basic (basicinput | basic_output | composite | variable) >
<! ATTLIST basic
     Integ Integ Integ Integ

<! ELEMENT basicinput (computation | input | selector) >
<! ATTLIST basicinput
     Integ CDATA #FIXED "basic_input" - -Integrator object- -
     Integ superclass #FIXED "basic"
     Integ inheritors #FIXED "computation input selector"
     Integ type #FIXED "none"

<! ELEMENT compute - O EMPTY >
<! ATTLIST compute
     Integ CDATA #FIXED "compute" - -Integrator object- -
     Integ superclass #FIXED "basic_input"
     Integ inheritors #FIXED "none"
     Integ type #FIXED "input"

- -attributes specific to Integrator- -
-include basic superclass attributes- -
- -include compont superclass attributes- -

- -attributes specific to Integrator- -
-include compont superclass attributes- -

- -attributes specific to Integrator- -
-include compont superclass attributes- -

- -other attributes- -
APPENDIX A. HYTIME LISTING OF OBJECTS

<table>
<thead>
<tr>
<th>id</th>
<th>ID</th>
<th>IMPLIED</th>
<th>-id of object-</th>
</tr>
</thead>
<tbody>
<tr>
<td>objid</td>
<td>IDREF</td>
<td>REQUIRED</td>
<td>-reference id for object-</td>
</tr>
<tr>
<td>name</td>
<td>NAME</td>
<td>&quot;def.anim&quot;</td>
<td>-default name of computation object-</td>
</tr>
<tr>
<td>fname</td>
<td>FName</td>
<td>&quot;comp.c&quot;</td>
<td>-filename of computation object-</td>
</tr>
<tr>
<td>orelation</td>
<td>SDATA</td>
<td>IMPLIED</td>
<td>-relation to other objects in IMMA-</td>
</tr>
<tr>
<td>track</td>
<td>NUMBER</td>
<td>IMPLIED</td>
<td>-track where computation is placed-</td>
</tr>
<tr>
<td>source</td>
<td>(harddisk</td>
<td>cdrom</td>
<td>floppy)</td>
</tr>
<tr>
<td>paramtrs</td>
<td>NMTOKENS</td>
<td>REQUIRED</td>
<td>-parameters passed to external code-</td>
</tr>
<tr>
<td>retvval</td>
<td>NMTOKENS</td>
<td>REQUIRED</td>
<td>-return values from external code-</td>
</tr>
<tr>
<td>memo</td>
<td>NMTOKENS</td>
<td>REQUIRED</td>
<td>-descriptive text about code-</td>
</tr>
</tbody>
</table>

- -include basic.input superclass attributes- -

> 

<! ELEMENT input -O (clock | key | mouse) >

<! ATTLIST input |

- -attributes specific to Integrator- |

Integ CDATA #FIXED "input" | -Integrator object- |

Integ superclass #FIXED "basic输入" |

Integ inheritors #FIXED "clock key mouse" |

Integ type #FIXED "input"

- -include basic.input superclass attributes- -

> 

<! ELEMENT clock -O EMPTY >

<! ATTLIST clock |

- -attributes specific to Integrator- |

Integ CDATA #FIXED "clock" | -Integrator object- |

Integ superclass #FIXED "input" |

Integ inheritors #FIXED "none" |

81
APPENDIX A. HYTIME LISTING OF OBJECTS

Integ  type #FIXED "input"
- -other attributes- -

id   ID #IMPLIED - -id of object- -
objid IDREF #REQUIRED - -reference id for object- -
name NAME "def_key" - -name of the clock object- -
timeout NUMBER #IMPLIED - -duration of timeout- -
track NUMBER #IMPLIED - -track where key effect takes place- -
orelation SDATA #IMPLIED - -relation to other objects in IMMA- -

- -include input superclass attributes- -
>

<! ELEMENT key (O EMPTY) >
<! ATTLIST key
- -attributes specific to Integrator- -
Integ CDATA #FIXED "key" - -Integrator object- -
Integ superclass #FIXED "input"
Integ inheritors #FIXED "none"
Integ type #FIXED "input"
- -other attributes- -
id ID #IMPLIED - -id of object- -
objid IDREF #REQUIRED - -reference id for object- -
name NAME "def_key" - -name of the key object- -
timeout NUMBER #IMPLIED "def_infinite" - -duration of timeout- -
track NUMBER #IMPLIED - -track where key effect takes place- -
orelation SDATA #IMPLIED - -relation to other objects in IMMA- -
action CDATA #REQUIRED - -action to be taken- -
keysel NMTOKEN #REQUIRED - -selected keys allowable- -

- -include input superclass attributes- -
>

<! ELEMENT Mouse (Input) object - ->
<! ELEMENT Usable Object - ->

82
APPENDIX A. HYTIME LISTING OF OBJECTS

<! ELEMENT mouse - O EMPTY >
<! ATTLIST mouse
- -attributes specific to Integrator- -
Integ CDATA #FIXED "mouse" - -Integrator object- -
Integ superclass #FIXED "input"
Integ inheritors #FIXED ""
Integ type #FIXED "input"
- -other attributes- -
id ID #IMPLIED - -id of object- -
objid IDREF #REQUIRED - -reference id for object- -
name NAME #FIXED "def_mouse" - -name of the mouse object- -
button (left | center | right) left - -mouse button selected- -
timeout NUMBER #IMPLIED - -duration of timeout- -
track NUMBER #IMPLIED - -track where mouse effect takes place- -
orelation SDATA #IMPLIED - -relation to other objects in IMMA- -
retval SDATA #REQUIRED - -return value of selected mouse action- -
- -include input superclass attributes- -
>

<! ELEMENT selector - O (hotspot | selkey | selmenu | selmouse) >
<! ATTLIST selector
- -attributes specific to Integrator- -
Integ CDATA #FIXED "selector" - -Integrator object- -
Integ superclass #FIXED "basic_input"
Integ inheritors #FIXED "hotspot selkey selmenu selmouse"
Integ type #FIXED "input_object"
- -include input object superclass attributes- -
>

<! ELEMENT HotSpot object - - >
<! ELEMENT Supportive Object - - >
### APPENDIX A: HYTIME LISTING OF OBJECTS

**<! ELEMENT hotspot - O EMPTY >**

**<! ATTLIST hotspot**

- -attributes specific to Integrator-

| Integ | CDATA | #FIXED | “hotspot” | -Integrator related object- |
| Integ | superclass | #FIXED | “selector” | -attributes specific to Integrator- |
| Integ | inheritors | #FIXED | “none” |
| Integ | type | #FIXED | “inputbject” |

- -other attributes-

| id | ID | #IMPLIED | -id of object- |
| length | NUMBER | #IMPLIED | -length of hotspot- |
| width | NUMBER | #IMPLIED | -width of hotspot- |
| transp | (yes | no) | yes | -hotspot transparent on screen- |
| action | CDATA | #REQUIRED | -action to be taken- |
| orelation | SDATA | #IMPLIED | -relation to other objects in IMMA- |

- -include selector superclass attributes-

**<! - - Selector Key object - - >**

**<! - - Usable Object - - >**

**<! ELEMENT selkey - O EMPTY >**

**<! ATTLIST selkey**

- -attributes specific to Integrator-

| Integ | CDATA | #FIXED | “selkey” | -Integrator object- |
| Integ | superclass | #FIXED | “selector” |
| Integ | inheritors | #FIXED | “none” |
| Integ | type | #FIXED | “inputbject” |

- -other attributes-

| id | ID | #IMPLIED | -id of object- |
| orelation | SDATA | #IMPLIED | -relation to other objects in IMMA- |
| action | CDATA | #REQUIRED | -action to be taken- |
| keysel | NMToken | #REQUIRED | -selected key- |
| retval | NMToken | #REQUIRED | -return value for key- |
| message | NMToken | #IMPLIED | -message to be displayed- |
| logicop | NAME | #IMPLIED | -logical operator- |

- -include selector superclass attributes-
APPENDIX A. HYTIME LISTING OF OBJECTS

> 

<! ELEMENT selmenu - C EMPTY > 
<! ATTLIST selmenu - -attributes specific to Integrator- - 
Integ CDATA #FIXED "selmenu" - -Integrator object- - 
Integ superclass #FIXED "selector" 
Integ inheritors #FIXED "none" 
Integ type #FIXED "input_object" 
- -other attributes- - 
id ID #IMPLIED - -id of object- - 
length NUMBER #IMPLIED - -length of menu button- - 
width NUMBER #IMPLIED - -width of menu button- - 
items NUMBER #IMPLIED - -number of menu items- - 
itemname NM_TOKENS #IMPLIED - -names of menu items- - 
menulen NUMBER #IMPLIED - -length of menu- - 
menuwid NUMBER #IMPLIED - -width of menu- - 
action CDATA #REQUIRED - -action to be taken- - 
hierar (yes | no) yes - -hierarchical (child) menu allowed- - 
- -include selector superclass attributes- - 
> 

<! ELEMENT selmouse - O EMPTY > 
<! ATTLIST selmouse - -attributes specific to Integrator- - 
Integ CDATA #FIXED "selmouse" - -Integrator object- - 
Integ superclass #FIXED "selector" 
Integ inheritors #FIXED "none" 
Integ type #FIXED "input_object" 
- -other attributes- - 
id ID #IMPLIED - -id of object- - 
orelation SDATA #IMPLIED - -relation to other objects in IMMA- - 
action CDATA #REQUIRED - -action to be taken- -
APPENDIX A. HYTIME LISTING OF OBJECTS

mousesel (left | right | mid) left
scancode SDATA #IMPLIED
retval NMTOKEN #REQUIRED
message NMTOKEN #IMPLIED
logicop NAME #IMPLIED

-include selector superclass attributes-

<! ELEMENT Integ Integer Integer Integer Integer>
<! ATTLIST Integ Integ Integer Integer II mid left SDATA #IMPLIED NMTOKEN #REQUIRED NMTOKEN #IMPLIED NAME #IMPLIED -include selector superclass attributes-

> 

<! ELEMENT basic_output - - (animation | audio | graphics | still | text | transition | video | window) -> 
<! ATTLIST basic_output 
basic...output - -attributes specific to Integrator- 
Integ CDATA #FIXED "basic...output" - -Integrator object- 
Integ superclass #FIXED "basic" 
Integ inheritors #FIXED "animation audio graphics still text transition video window" 
Integ type #FIXED "none" 

-include basic superclass attributes-

> 

<! ELEMENT animate - O EMPTY -> 
<! ATTLIST animate 
animate - -attributes specific to Integrator- 
Integ CDATA #FIXED "animate" - -Integrator object- 
Integ superclass #FIXED "basic_output" 
Integ inheritors #FIXED "none" 
Integ type #FIXED "output_object" 

- -other attributes-

id ID #IMPLIED - -id of object-
objid IDREF #REQUIRED - -reference id for object-
name NAME "def_anim" - -name of animation object- 

86
APPENDIX A. HYTIME LISTING OF OBJECTS

```xml
<ELEMENT audio start,end=0 end=end EMPTY=
<ATTLIST id ID #IMPLIED "id of object" -
<ATTLIST objid IDREF #REQUIRED "reference id for object" -
<ATTLIST name NAME #REQUIRED "name of still object" -
<ATTLIST fname FNAME #REQUIRED "filename of still" -
<ATTLIST timespec (start,end | start,dur | dur,end) start,end=0 end=end duration=90 "time specification as pair of numbers representing start, duration or end time" -
<ATTLIST durframe (start, end) 0 90 "duration for audio (frames)" -
<ATTLIST volume (right | left | both) both "volume control for audio" -
<ATTLIST speed NMTOKEN #IMPLIED "speed for audio sequence" -
<ATTLIST relation SDATA #IMPLIED "relation to other objects in IMMA" -
<ATTLIST channel (right | left | both) both "channel control for audio" -
<ATTLIST source (laudisk | videodisc | floppy) hardisk "source of audio" -
<ATTLIST fmt (MPEG | DVI | CD-I) DVI "file format of audio" -
<ATTLIST nyquist NUMBER #IMPLIED "nyquist frequency in Hz" -
<ATTLIST filter NUMBER #IMPLIED "filter cut off frequency" -
<ATTLIST attenu NUMBER #IMPLIED "volume in decibels" -
<ATTLIST fframe #IMPLIED "frame rate of audio" -
<ATTLIST type (stereo | mono) stereo "audio format mono or stereo" -
```

87
APPENDIX A. HYTIME LISTING OF OBJECTS

- include basic_output superclass attributes -

> 

<!- - Graphics object - ->
<!- - Usable Object - ->

<!- ELEMENT graphics - O EMPTY ->
<!- ATTLIST graphics - attributes specific to Integrator - ->
Integ CDATA #FIXED "graphics" - Integrator object -
Integ superclass #FIXED "basic_output" -
Integ inheritors #FIXED "none" -
Integ type #FIXED "output_object" -

- other attributes -
id ID #IMPLIED - id of object -
objid IDREF #REQUIRED - reference id for object -
name NAME #REQUIRED "def_grap" - name of graphics object -
timespec (start,end | start,dur | dur,end) start,end - time specification as pair of numbers representing start, duration or end time -
relation SDATA #IMPLIED - relation to other objects in IMMA -
track NUMBER #IMPLIED - track where still is placed -
exported SDATA #IMPLIED - is graphics exported into IMMA -
moveable (yes | no) no - can presentation be moved -
resize (yes | no) no - can presentation be resized -

- include basic_output superclass attributes -

> 

<!- - Still Image object - ->
<!- - Usable Object - ->

<!- ELEMENT still - O EMPTY ->
<!- ATTLIST still - attributes specific to Integrator - ->
Integ CDATA #FIXED "stills" - Integrator object -
Integ superclass #FIXED "basic_output" -
Integ inheritors #FIXED "none" -
Integ type #FIXED "output_object" -

- other attributes -
APPENDIX A. HYTIME LISTING OF OBJECTS

id   ID     IMPLIED   - id of object -
objid IDREF #REQUIRED - reference id for object -
name NAME "def.still" - name of still object -
fname FNAME "samp.c18" - filename of still -
xres NUMBER #IMPLIED - xresolution of still -
yres NUMBER #IMPLIED - yresolution of still -
cmpfrmt (uncomp | comp) uncomp - compression format of the still -
bpp NUMBER #IMPLIED - bits per pixel of still -
compval NUMBER #IMPLIED - compression value of the still -
timespec (start,end | start,dur | dur,end) start,end - time specification as pair of numbers representing start, duration or end time -
source (harddisk | videodisc | floppy) harddisk - source of the still image -
filefmt (JPEG | DVI | PostScript) DVI - file format; add others as needed -
orelation SDATA #IMPLIED - relation to other objects in IMMA -
hotspot NUTOKENS #IMPLIED - hotspots defined on the still -
track NUMBER #IMPLIED - track where still is placed -

- include basic..output superclass attributes -

<! ELEMENT text EMPTY >
<! ELEMENT text - - Usable Object - - >
<! ATTLIST text
  id ID #IMPLIED - id of object -
  objid IDREF #REQUIRED - reference id for object -
  name NAME "def.text" - name of text object -
  timespec (start,end | start,dur | dur,end) start,end - time specification as pair of numbers representing start, duration or end time -
  message NMTOKEN #IMPLIED - text message to be displayed -
  fonttype NUTOKENS #IMPLIED - type of font -
  fontsize NUTOKENS #IMPLIED - size of font -
  fontcolr NUTOKENS #IMPLIED - color of font -

89
APPENDIX A. HYTIME LISTING OF OBJECTS

shadow (yes | no) no - -shadow enabled or disabled- -
outline (yes | no) no - -outline enabled or disabled- -
bkcolor NUTOKEN #IMPLIED - -background color- -
relation SDATA #IMPLIED - -relation to other objects in IMMA- -
track NUMBER #IMPLIED - -track where still is placed- -

- -include basic_output superclass attributes- -

<!- - Transition object - ->
<!- - Supportive Object - ->

<! ELEMENT transition - O (audtrans | stiltrans | vidtrans) >
<! ATTLIST transition
- -attributes specific to Integrator- -
Integ CDATA #FIXED "transition" - -Integrator object- -
Integ superclass #FIXED "basic_output"
Integ inheritors #FIXED "audtrans stiltrans vidtrans"
Integ type #FIXED "output_object"

- -include basic_output superclass attributes- -

<!- - Audio transition object - ->
<!- - Usable Object - ->

<! ELEMENT audtrans - O EMPTY >
<! ATTLIST audtrans
- -attributes specific to Integrator- -
Integ CDATA #FIXED "audtrans" - -Integrator object- -
Integ superclass #FIXED "transition"
Integ inheritors #FIXED "none"
Integ type #FIXED "output_object"

- -other attributes- -
id ID #IMPLIED - -id of object- -
objid IDREF #REQUIRED - -reference id for object- -
name NAME "def.astrans" - -filename of the transition object- -
type NMTOKEN "fade.both" - -type of transition - -
timespec (start,end | start,dur | dur,end) start,end - -time specification as pair of numbers
APPENDIX A. HYTIME LISTING OF OBJECTS

plcement (before | after)  
  - transition position wrt to modified object -

speed NMTOKEN  #IMPLIED  
  - speed for transition effect -

modified IDREF  #REQUIRED  
  - id reference to modified object -

orelation SDATA  #IMPLIED  
  - relation to other objects in IMMA -

- -include transition superclass attributes- -
>

<!- - Still transition object - ->
<!- - Usable Object - ->

<! ELEMENT stltrans - O EMPTY >
<! ATTLIST stltrans
  - attributes specific to Integrator- -

Integ CDATA  #FIXED  "stltrans"  
  - Integrator object -

Integ superclass  #FIXED  "transition"  

Integ inheritors  #FIXED  "none"  

Integ type  #FIXED  "output_object"  
  - other attributes -

id ID  #IMPLIED  
  - id of object -

objid IDREF  #REQUIRED  
  - reference id for object -

name NAME  "def_strans"  
  - filename of the transition object -

type NMTOKEN  "wipe_right"  
  - type of transition -

timespec (start,end | start,dur | dur,end)  
  - time specification as pair of numbers representing start, duration or end time -

plcement (before | after)  
  - transition position wrt to modified object -

speed NMTOKEN  #IMPLIED  
  - speed for transition effect -

track NUMBER  #IMPLIED  
  - track where transition effect is placed -

modified IDREF  #REQUIRED  
  - id reference to modified object -

orelation SDATA  #IMPLIED  
  - relation to other objects in IMMA -

- -include transition superclass attributes- -
>

<!- - Video transition object - ->
<!- - Usable Object - ->

<! ELEMENT vidtrans - O EMPTY >
<! ATTLIST vidtrans

91
APPENDIX A. HYTIME LISTING OF OBJECTS

- attributes specific to Integrator -

IntegCDATA #FIXED "vidtrans" - Integrator object -
Integsuperclass #FIXED "transition"
Integinheritors #FIXED "none"
Integtype #FIXED "output_object"

- other attributes -

idID #IMPLIED - id of object -
objidIDREF #REQUIRED - reference id for object -
nameNAME "def_vtrans" - filename of the transition object -
typeNMTOKEN "fade" - type of transition -
timespec (start, end | start, dur | dur, end) start, end - time specification as pair of numbers representing start, duration or end time -
placement (before | after) after - transition position wrt to modified object -
speedNMTOKEN #IMPLIED - speed for transition effect -
trackNUMBER #IMPLIED - track where transition effect is placed -
modifiedIDREF #REQUIRED - id reference to modified object -
relationSDATA #IMPLIED - relation to other objects in IMMA -
paramsNMTOKENS #REQUIRED - parameters passed to external code -

- include transition superclass attributes -

> 

<! - Video object - >
<! - Usable Object - >
<! ELEMENT video - O EMPTY >
<! ATTLIST video - attributes specific to Integrator -

IntegCDATA #FIXED "video" - Integrator object -
Integsuperclass #FIXED "basic_output"
Integinheritors #FIXED "none"
Integtype #FIXED "output_object"

- other attributes -

idID #IMPLIED - id of object -
objidIDREF #REQUIRED - reference id for object -
nameNAME "def_video" - name of video object -
fnameFNAME "samp.avs" - filename of video -
xresNUMBER #IMPLIED - xresolution of video -
yresNUMBER #IMPLIED - yresolution of video -
speedNUTOKEN #IMPLIED - speed of video -
APPENDIX A. HYTIME LISTING OF OBJECTS

compval NUMBER #IMPLIED - compression value of video -
compress NAME #IMPLIED - compression format of video -
durframe (start, end) 0 90 - duration for video (frames) -
timespec (start,end | start,dur | dur,end) start,end - time specification as pair of numbers representing start, duration or end time -
source (harddisk | videodisc | vcr) harddisk - source of the video -
filefmt (JPEG | DVI | NTSC) DVI - file format of the video -
orientation SDATA #IMPLIED - relation to other objects in IMMA -
hotspot NUTOKENS #IMPLIED - hotspots defined on the video -
track NUMBER #IMPLIED - track where video is placed -

- include basic_output superclass attributes -

> 

<! - Window object - ->
<! - Usable Object - ->
<! ELEMENT window - O EMPTY >
<! ATTLIST window - -attributes specific to Integrator- -
Integ CDATA #FIXED "window" - Integrator object -
Integ superclass #FIXED "basic_output" -
lateg inheritors #FIXED "none" -
Integ type #FIXED "output_object" -
- other attributes -

id ID #IMPLIED - id of object -
objid IDREF #REQUIRED - reference id for object -
length NUMBER #IMPLIED - length of window -
width NUMBER #IMPLIED - width of window -
title NM_TOKEN #IMPLIED - title strings -
scroll (yes | no) yes - display scroll bars -
movable (yes | no) no - can window be moved -
resize (yes | no) no - can window be resized -
dspmxbtn (yes | no) yes - display maximize button -
dspmnbtn (yes | no) yes - display minimize button -
dspahbtn (yes | no) yes - display help button -
dspwbbtn (yes | no) yes - display window button -
pane (yes | no) yes - display pane -
pushpin (yes | no) yes - display pushpin button -
raisewin (yes | no) yes - display bring window to front button -

93
APPENDIX A. HYTIME LISTING OF OBJECTS

lowerwin (yes | no) yes -display push window to back button-
refresh (yes | no) yes -refresh window-
footer (yes | no) yes -display status messages in footer-
forecolor NMTOKEN #IMPLIED -foreground color-
bkcolor NMTOKEN #IMPLIED -background color-
menubtn NUMBER #IMPLIED -set number of menu buttons-
menuatr NUMBER #IMPLIED -set attributes for menu buttons-
border NUMBER #IMPLIED -set border width for window-
cursor (arrow | text | cross) arrow -set type of cursor-
font NUTOKEN #IMPLIED -set font type for window-
inistate (base | popup | icon) base -initial state of window-

-include basic output superclass attributes-

<!--
Composite object -->
<!-Usable Object -->

<! ELEMENT composite - - (sce)> -“special core event” defined below-
<! ATTLIST composite
- - -attributes specific to Integrator-
Integ CDATA #FIXED “composite” -Integrator object-
Integ superclass #FIXED “basic”
Integ inheritors #FIXED “compin compintr compout”
Integ type #FIXED “none”

-include basic superclass attributes-

<! ELEMENT sce - - (compin | compintr | compout) -see is either a compin, compintr, or compout element-
<! ATTLIST scein - - (ce) -
<! ELEMENT cein - - (ce) -
<! ATTLIST cein
Integ type #FIXED “input” -must be input type objects only-

<! ELEMENT ceintr - - (ce) -
<! ATTLIST ceintr
Integ type #FIXED “interactive” -both input and output type-

94
APPENDIX A. HYTIME LISTING OF OBJECTS

> <! ELEMENT cout - - (ce) >
<! ATTLIST cein
Integ type #FIXED "output" - -must be output type objects only- -
>
- -ce, evseq, and ces are HyTime events defined in the DIS- -
<! - - Composite Input object - - >
<! - - Usable Object - - >

<! ELEMENT compin - - (cein | evseq | ces) >
<! ATTLIST compin
Integ CDATA #FIXED "compin" - -Integrator object- -
Integ superclass #FIXED "composite"
Integ inheritors #FIXED combination of basic_input objects
Integ type #FIXED "input"
- -other attributes- -
id ID #IMPLIED - -id of object- -
objid IDREF #REQUIRED - -reference id for object- -
name NAME "def_compin" - -name of composite input object- -
orelation SDATA #IMPLIED - -relation to other objects in IMMA- -
track NUMBER #IMPLIED - -track where composite input is placed- -
orelattr SDATA #IMPLIED - -relation to objects within composite- -
objlist SDATA #IMPLIED - -list of objects within composite- -

- -include composite superclass attributes- -
>

<! - - Composite Interactive object - - >
<! - - Usable Object - - >

<! ELEMENT compintr - - (ceintr | evseq | ces) >
<! ATTLIST compintr
Integ CDATA #FIXED "compintr" - -Integrator object- -
Integ superclass #FIXED "composite"
Integ inheritors #FIXED combination of basic_input & basic_output objects
APPENDIX A. HYTIME LISTING OF OBJECTS

Integ type #FIXED "interactive"
   - other attributes -
   id ID #IMPLIED - -id of object- -
objid IDREF #REQUIRED - -reference id for object- -
name NAME "def.compintr" - -name of composite interactive object- -
orelation SDATA #IMPLIED - -relation to other objects in IMMA- -
track NUMBER #IMPLIED - -track where composite interactive object is placed- -
orelattr SDATA #IMPLIED - -relation to objects within composite- -
objlist SDATA #IMPLIED - -list of objects within composite- -

- include composite superclass attributes -
>

<! ELEMENT compout - - (ceout | evseq | ces) - ->
<! ATTLIST compout
- -attributes specific to Integrator- -
Integ CDATA #FIXED "compout" - -Integrator object- -
Integ superclass #FIXED "composite" - -
Integ inherits #FIXED combination of basic output objects - -
Integ type #FIXED "output" - -
   - other attributes -
   id ID #IMPLIED - -id of object- -
objid IDREF #REQUIRED - -reference id for object- -
name NAME "def.compout" - -name of composite output object- -
orelation SDATA #IMPLIED - -relation to other objects in IMMA- -
track NUMBER #IMPLIED - -track where composite output object is placed- -
orelattr SDATA #IMPLIED - -relation to objects within composite- -
objlist SDATA #IMPLIED - -list of objects within composite- -

- include composite superclass attributes -
>

<! ELEMENT Variable object - ->

96
APPENDIX A. HYTIME LISTING OF OBJECTS

<! ELEMENT variable - O EMPTY >
<! ATTLIST variable
  id #IMPLIED
  objid #REQUIRED
  name #IMPLIED
  memo #IMPLIED
  length #IMPLIED
  type #IMPLIED
  scope #IMPLIED
  reserved (yes | no) #IMPLIED

  <!- -Integrator object- ->
  Integ CDATA #FIXED "variable" -Integrator object- -
  Integ superclass #FIXED "none"
  Integ inheritors #FIXED "none"
  Integ type #FIXED "none"
  - -other attributes- -
  id ID #IMPLIED -id of object- -
  objid IDREF #REQUIRED -reference id for object- -
  name NAME "def_var" -name of variable- -
  memo NMTOKENS #IMPLIED -memo attached with variable- -
  length NUMBER #FIXED 32 -length of variable- -
  type (int | float | double) int -type of variable- -
  scope NAME #IMPLIED -scope of variable- -
  reserved (yes | no) no -reserved variable or user defined- -

>

<! ELEMENT control - - (baton | conditional | tie) >
<! ATTLIST control
  id #IMPLIED
  objid #REQUIRED
  name #IMPLIED
  memo #IMPLIED
  length #IMPLIED
  type #IMPLIED
  scope #IMPLIED

  <!- -Integrator object- ->
  Integ CDATA #FIXED "control" -Integrator object- -
  Integ superclass #FIXED "compont"
  Integ inheritors #FIXED "baton conditional tie"
  Integ type #FIXED "control_object"
  - -include compont superclass attributes- -

>

<! ELEMENT baton - - (proscope)+ >
<! ATTLIST baton
  id #IMPLIED
  objid #REQUIRED
  name #IMPLIED

  <!- -Usable Object- ->
  Integ CDATA #FIXED "baton" -Usable Object- -

>

97
APPENDIX A. HYTIME LISTING OF OBJECTS

- attributes specific to Integrator -

Integ CDATA #FIXED "baton" - Integrator object -
Integ superclass #FIXED "control"
Integ inheritors #FIXED "master slave"
Integ type #FIXED "controLobject"

- other attributes -

id ID #REQUIRED - id of object -
objid IDREF #REQUIRED - reference id for object -
trackid IDREF #REQUIRED - id of track controlled by this object -
name NAME "mstr_btn" - name of the baton object -
timespec (start, stop, increment) 0 10 ! - time specification for baton in terms of start, end and increment time -
rated NMTOKEN #IMPLIED - rate at which time is changed -
defined (yes | no) no - whether application open ended or not -
random (yes | no) no - timing of baton random or not -
mode (regular | input_converted) regular - baton has regular or input_converted clock pulse -
tempo NUTOKENS #CURRENT - tempos defined in baton -

- include control superclass attributes -
- include HyTime baton attributes -

proscope is a HyTime element defined in the DIS -

<!ELEMENT slave (slavescr | slavetrk) >
<! ATTLIST slave

- attributes specific to Integrator -

Integ CDATA #FIXED "slave" - Integrator object -
Integ superclass #FIXED "baton"
Integ inheritors #FIXED "slavescr slavetrk"
Integ type #FIXED "controLobject"

- include baton superclass attributes -

>
APPENDIX A. HYTIME LISTING OF OBJECTS

<! ELEMENT slavescr - O EMPTY >
<! ATTLIST slavescr
  id CDATA #FIXED "slavescr" - -attributes specific to Integrator- -
  superclass CDATA #FIXED "slave" - Integrator object- -
  inheritors CDATA #FIXED "none" - - -
  type CDATA #FIXED "control_object" - - other attributes- -
  ID IDREF #IMPLIED - id of object- -
  objid IDREF #REQUIRED - reference id for object- -
  name NAME "slave_scr" - -name of the slave score object- -

- include slave superclass attributes- -
- include HyTime baton attributes- -

<! ELEMENT slavetrk - O EMPTY >
<! ATTLIST slavetrk
  id CDATA #IMPLIED - -id of object- -
  objid IDREF #REQUIRED - -reference id for object- -
  name NAME "slave_trk" - -name of the slave track object- -

- include slave superclass attributes- -
- include HyTime baton attributes- -

<! ELEMENT Conditional object - ->
<! ELEMENT Supportive Object - ->
APPENDIX A. HYTIME LISTING OF OBJECTS

<! ELEMENT conditional "-(dowhile | forloop | go | iftheloop | loopn | onxdo | rptuntil | while) »>

<! ATTLIST conditional
-attributes specific to Integrator-

Integ cdatta #FIXED "conditional" -Integrator object-
Integ superclass #FIXED "control"
Integ inheritors #FIXED "dowhile forloop goto iftheloop loopn onxdo rptuntil while"
Integ type #FIXED "controlObject"

-include control superclass attributes-

<! ELEMENT action "-(#PCDATA) »>
<! ELEMENT cond "-(#PCDATA) »>

<!- Do While object - »>
<!- Usable Object - »>

<! ELEMENT dowhile "-(action', cond) »>
<! ATTLIST dowhile
-attributes specific to Integrator-

Integ pdatta #FIXED "dowhile" -Integrator object-
Integ superclass #FIXED "conditional"
Integ inheritors #FIXED "none"
Integ type #FIXED "controlObject"

-other attributes-

id ID #IMPLIED -id of object-
objid IDREF #REQUIRED -reference id for object-

-include conditional superclass attributes-

<! ELEMENT forloop "-(init, final, index, action) »>
<! ATTLIST forloop
-attributes specific to Integrator-

100
APPENDIX A. HYTIME LISTING OF OBJECTS

<table>
<thead>
<tr>
<th>Integ</th>
<th>CDATA</th>
<th>#FIXED</th>
<th>&quot;forloop&quot;</th>
<th>- Integrator object -</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integ</td>
<td>superclass</td>
<td>#FIXED</td>
<td>&quot;conditional&quot;</td>
<td></td>
</tr>
<tr>
<td>Integ</td>
<td>inheritors</td>
<td>#FIXED</td>
<td>&quot;none&quot;</td>
<td></td>
</tr>
<tr>
<td>Integ</td>
<td>type</td>
<td>#FIXED</td>
<td>&quot;controLobject&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>id</td>
<td>ID</td>
<td>IMPLIED</td>
<td>- id of object -</td>
</tr>
<tr>
<td></td>
<td>objid</td>
<td>IDREF</td>
<td>#REQUIRED</td>
<td>- reference id for object -</td>
</tr>
</tbody>
</table>

- Include conditional superclass attributes -

> <! ELEMENT init - - (#PCDATA) > - initial condition -
<! ELEMENT final - - (#PCDATA) > - final condition -
<! ELEMENT index - - (#PCDATA) > - step action -

<! ELEMENT goto - O EMPTY >
<! ATTLIST goto - - attributes specific to Integrator -
<table>
<thead>
<tr>
<th>Integ</th>
<th>CDATA</th>
<th>#FIXED</th>
<th>&quot;goto&quot;</th>
<th>- Integrator object -</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integ</td>
<td>superclass</td>
<td>#FIXED</td>
<td>&quot;conditional&quot;</td>
<td></td>
</tr>
<tr>
<td>Integ</td>
<td>inheritors</td>
<td>#FIXED</td>
<td>&quot;none&quot;</td>
<td></td>
</tr>
<tr>
<td>Integ</td>
<td>type</td>
<td>#FIXED</td>
<td>&quot;controLobject&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>id</td>
<td>ID</td>
<td>IMPLIED</td>
<td>- id of object -</td>
</tr>
<tr>
<td></td>
<td>objid</td>
<td>IDREF</td>
<td>#REQUIRED</td>
<td>- reference id for object -</td>
</tr>
<tr>
<td></td>
<td>desi</td>
<td>NAME</td>
<td>#REQUIRED</td>
<td>- name of destination -</td>
</tr>
<tr>
<td></td>
<td>destid</td>
<td>IDREF</td>
<td>#REQUIRED</td>
<td>- id of destination -</td>
</tr>
<tr>
<td></td>
<td>displink</td>
<td>(yes</td>
<td>no) no</td>
<td>- display link to target in Integrator -</td>
</tr>
</tbody>
</table>

- Include conditional superclass attributes -

> <! ELEMENT ifthel - O (cond, then, else?) >
<! ATTLIST ifthel - - attributes specific to Integrator -

101
APPENDIX A. HYTIME LISTING OF OBJECTS

Integ CDATA #FIXED "ifthen" - Integrator object -
Integ superclass #FIXED "conditional"
Integ inheritors #FIXED "none"
Integ type #FIXED "controlObject"

- other attributes -

id ID #IMPLIED - id of object -
objid IDREF #REQUIRED - reference id for object -

- include conditional superclass attributes -

>
<! ELEMENT then - - (#PCDATA) >
<! ELEMENT else - - (#PCDATA) >

<! - - Loop N object - - >
<! - - Usable Object - - >

<! ELEMENT loopn - - ((cond | var) & statement+) >
<! ATTLIST loopn
- attributes specific to Integrator -

Integ CDATA #FIXED "loopn" - Integrator object -
Integ superclass #FIXED "conditional"
Integ inheritors #FIXED "none"
Integ type #FIXED "controlObject"

- other attributes -

id ID #IMPLIED - id of object -
objid IDREF #REQUIRED - reference id for object -

- include conditional superclass attributes -

>
<! ELEMENT lcond - - (#PCDATA) >
<! ATTLIST lcond - - iterate NUMBER #CONREF >
<! ELEMENT var - - (#PCDATA) >
<! ELEMENT statement - - (#PCDATA) >

<! - - On X Do object - - >
<! - - Usable Object - - >
APPENDIX A. HYTIME LISTING OF OBJECTS

<! ELEMENT onxdo - - (X, do) >
<! ATTLIST onxdo
  -attributes specific to Integrator- -
  Integ CDATA #FIXED "onxdo" - -Integrator object- -
  Integ superclass #FIXED "conditional"
  Integ inherits #FIXED "none"
  Integ type #FIXED "controlObject"
  -other attributes-
  id ID #IMPLIED - -id of object- -
  objid IDREF #REQUIRED - -reference id for object- -

-include conditional superclass attributes-

>
<! ELEMENT X - - (#PCDATA)> - -condition element- -
<! ELEMENT do - - (#PCDATA)> - -action element- -

<!- - Repeat Until object - ->
<!- - Usable Object - ->

<! ELEMENT rptuntil - - (repeat, until) >
<! ATTLIST rptuntil
  -attributes specific to Integrator- -
  Integ CDATA #FIXED "rptuntil" - -Integrator object- -
  Integ superclass #FIXED "conditional"
  Integ inherits #FIXED "none"
  Integ type #FIXED "controlObject"
  -other attributes-
  id ID #IMPLIED - -id of object- -

-include conditional superclass attributes-

>
<! ELEMENT repeat - - (#PCDATA)> - -repeat condition- -
<! ELEMENT until - - (#PCDATA)> - -until condition- -

<!- - While object - ->
<!- - Usable Object - ->

103
APPENDIX A. HYTIME LISTING OF OBJECTS

<! ELEMENT while (cond. action*) >
<! ATTLIST while
- -attributes specific to Integrator-
Integ CDATA #FIXED "while" - -Integrator object-
Integ superclass #FIXED "conditional"
Integ inheritors #FIXED "none"
Integ type #FIXED "controlObject"
- -other attributes-
id ID #IMPLIED - -id of object-
objid IDREF #REQUIRED - -reference id for object-

- -include conditional superclass attributes-

<!- -
<!- -
tie
- -attributes specific to Integrator-
Integ CDATA #FIXED "tie" - -Integrator object-
Integ superclass #FIXED "control"
Integ inheritors #FIXED "none"
Integ type #FIXED "controlObject"
- -other attributes-
id ID #IMPLIED - -id of object-
objid IDREF #REQUIRED - -reference id for object-
name NAME "def.tie" - -name of the slave baton object-
tielist NMtokens #REQUIRED - -list of tied objects-
tiefirst NUTOKEN #IMPLIED - -time from start of application of object at first end of tie-
tielast NUTOKEN #IMPLIED - -time from start of application of object at other end of tie-
disptie (yes | no) yes - -display tie-

- -include control superclass attributes-

->
<! ELEMENT tstart (tstart, tend) >
<! ATTLIST tstart
- -attributes specific to Integrator-
Integ CDATA #FIXED "tstart" - -Integrator object-
Integ superclass #FIXED "control"
Integ inheritors #FIXED "none"
Integ type #FIXED "controlObject"
- -other attributes-
objid IDREF #REQUIRED - -reference id for object-
name NAME "de$tstart" - -name of the slave baton object-
tielist NMtokens #REQUIRED - -list of tied objects-
tiefirst NUTOKEN #IMPLIED - -time from start of application of object at first end of tie-
tielast NUTOKEN #IMPLIED - -time from start of application of object at other end of tie-
disptie (yes | no) yes - -display tie-

- -include control superclass attributes-

->
APPENDIX A. HYTIME LISTING OF OBJECTS

<! ELEMENT tend - - (#PCDATA) >

<!- - Logical object - ->
<!- - Supportive Object - ->

<! ELEMENT logical - O (help | option | pool) >
<! ATTLIST logical

- -attributes specific to Integrator- -
Integ CDATA #FIXED "logical" - -Integrator object- -
Integ superclass #FIXED "compoint"
Integ inheritors #FIXED "help option pool"
Integ type #FIXED "logicalObject"

- -include compoint superclass attributes- -

->

<!- - Help object - ->
<!- - Usable Object - ->
<! ELEMENT help - O EMPTY >
<! ATTLIST help

- -attributes specific to Integrator- -
Integ CDATA #FIXED "help" - -Integrator object- -
Integ superclass #FIXED "logical"
Integ inheritors #FIXED "none"
Integ type #FIXED "logicalObject"

- -other attributes- -
id ID #IMPLIED - -id of object- -
objid IDREF #REQUIRED - -reference id for object- -
name NAME "def_help" - -name of the slave baton object- -
relation SDATA #IMPLIED - -relation to other objects in IMMA- -
track NUMBER #IMPLIED - -track where help is placed- -
file FILE #CONREF - -text or image file- -

- -include logical superclass attributes- -

->

<!- - Option object - ->
APPENDIX A. HYTIME LISTING OF OBJECTS

<!- - Supportive Object - ->

<! ELEMENT option - - {execute | presobj | prevlvl} >
<!ATTLIST option
- -attributes specific to Integrator-
Integ CDATA #FIXED "option" - -Integrator object- -
Integ superclass #FIXED "logical"
Integ inheritors #FIXED "execute presobj prevlvl"
Integ type #FIXED "logicalObject" - -include logical superclass attributes- -

<!- - Execute object - ->

<! ELEMENT execute - O EMPTY >
<! ATTLIST execute
- -attributes specific to Integrator-
Integ CDATA #FIXED "execute" - -Integrator object- -
Integ superclass #FIXED "option"
Integ inheritors #FIXED "none"
Integ type #FIXED "logicalObject" - -include option superclass attributes- -

<!- - Present Object object - ->

<! ELEMENT presobj - C EMPTY >
<! ATTLIST presobj
- -attributes specific to Integrator-
Integ CDATA #FIXED "presobj" - -Integrator object- -
Integ superclass #FIXED "option"
Integ inheritors #FIXED "none"
Integ type #FIXED "logicalObject" - -include option superclass attributes- -
APPENDIX A. HYTIME LISTING OF OBJECTS

<! ELEMENT prevlevl - O EMPTY >
<! ATTLIST prevlevl - -attributes specific to Integrator- -
Integ CDATA #FIXED "prevlevl" - Integrator object-
Integ superclass #FIXED "option"
Integ inheritors #FIXED "none"
Integ type #FIXED "logical_object"

- -include option superclass attributes- -

<! ELEMENT pool - - (by_name | by_type) >
<! ATTLIST pool - -attributes specific to Integrator- -
Integ CDATA #FIXED "pool" - Integrator object-
Integ superclass #FIXED "logical"
Integ inheritors #FIXED "by_name by_type"
Integ type #FIXED "logical_object"

- -other attributes- -
id ID #IMPLIED - id of object-
objid IDREF #REQUIRED - reference id for object-
length NUMBER #IMPLIED - length of window-
width NUMBER #IMPLIED - width of window-
title NMTOKEN #FIXED "POOL" - title string-
scroll (yes | no) yes - display scroll bars-
forecolor NMTOKEN #IMPLIED - foreground color-
bkgcolor NMTOKEN #IMPLIED - background color-
border NUMBER #IMPLIED - set border width for pool window-
moveable (yes | no) no - can pool window be moved-
resize (yes | no) no - can pool window be resized-
dspmxbtn (yes | no) yes - display maximize button-
dspmnbtn (yes | no) yes - display minimize button-
### APPENDIX A. HYTIME LISTING OF OBJECTS

- display help button -
- display window button -
- set type of cursor to arrow -
- set font type for window -
- display pushpin button -
- display bring window to front button -
- display push window to back button -
- refresh window -
- display status messages in footer -
- initial state of pool window -

- include logical superclass attributes -

```xml
<!ELEMENT ![Integ]
<!ATTLIST ![Integ]
```

- include pool superclass attributes -

```xml
<!ELEMENT ![Integ]
<!ATTLIST ![Integ]
```

- include pool superclass attributes -

```xml
<!ELEMENT ![Integ]
<!ATTLIST ![Integ]
```

- include pool superclass attributes -

```xml
<!ELEMENT ![Integ]
<!ATTLIST ![Integ]
```

- include pool superclass attributes -

```xml
<!ELEMENT ![Integ]
<!ATTLIST ![Integ]
```

- include pool superclass attributes -

```xml
<!ELEMENT ![Integ]
<!ATTLIST ![Integ]
```

- include pool superclass attributes -

```xml
<!ELEMENT ![Integ]
<!ATTLIST ![Integ]
```

- include pool superclass attributes -

```xml
<!ELEMENT ![Integ]
<!ATTLIST ![Integ]
```

- include pool superclass attributes -

```xml
<!ELEMENT ![Integ]
<!ATTLIST ![Integ]
```

- include pool superclass attributes -

```xml
<!ELEMENT ![Integ]
<!ATTLIST ![Integ]
```

- include pool superclass attributes -

```xml
<!ELEMENT ![Integ]
<!ATTLIST ![Integ]
```

- include pool superclass attributes -

```xml
<!ELEMENT ![Integ]
<!ATTLIST ![Integ]
```

- include pool superclass attributes -

```xml
<!ELEMENT ![Integ]
<!ATTLIST ![Integ]
```

- include pool superclass attributes -

```xml
<!ELEMENT ![Integ]
<!ATTLIST ![Integ]
```

- include pool superclass attributes -

```xml
<!ELEMENT ![Integ]
<!ATTLIST ![Integ]
```

- include pool superclass attributes -

```xml
<!ELEMENT ![Integ]
<!ATTLIST ![Integ]
```

- include pool superclass attributes -

```xml
<!ELEMENT ![Integ]
<!ATTLIST ![Integ]
```

- include pool superclass attributes -

```xml
<!ELEMENT ![Integ]
<!ATTLIST ![Integ]
```

- include pool superclass attributes -

```xml
<!ELEMENT ![Integ]
<!ATTLIST ![Integ]
```

- include pool superclass attributes -

```xml
<!ELEMENT ![Integ]
<!ATTLIST ![Integ]
```

- include pool superclass attributes -

```xml
<!ELEMENT ![Integ]
<!ATTLIST ![Integ]
```

- include pool superclass attributes -
APPENDIX A. HYTIME LISTING OF OBJECTS

> 

<? Element relational - (link | synchro) >

<? Element Supportive Object - ?>

<? ATTLIST relational
-attributes specific to Integrator-
Integ CDATA #FIXED "relational" - -Integrator object- -
Integ superclass #FIXED "compon"
Integ inheritors #FIXED "link synchro"
Integ type #FIXED "relation_object"

-include compon superclass attributes-

>

<? Element Link object - ?>

<? Element Usable Object - ?>

<? ATTLIST link
-attributes specific to Integrator-
Integ CDATA #FIXED "link" - -Integrator object- -
Integ superclass #FIXED "relational"
Integ inheritors #FIXED "none"
Integ type #FIXED "relation_object"

-other attributes-

id ID #IMPLIED - -id of object- -
objid IDREF #REQUIRED - -reference id for object- -
name NAME #REQUIRED "def.link" - -name of link object- -
memo NMTOKENS #IMPLIED - -memo attached with link- -
source NMTOKENS #REQUIRED - -source anchor of link- -
sink NMTOKENS #REQUIRED - -destination anchor of link- -
flow (data | control) data - -control or data flow- -
binding (static | dynamic) static - -control or data flow- -
display (yes | no) no - -display the link- -
scopetr (intra | inter) intra - -scope of link over track- -
scopescr (intra | inter) intra - -scope of link over score- -
APPENDIX A. HYTIME LISTING OF OBJECTS

- include relational superclass attributes -
- include HyTime link attributes -

<!- -
<!- Synchro object
<!- Usable Object
<! ELEMENT synchro ANY >
<! ATTLIST synchro
  Integ CDATA #FIXED "synchro" - Integrator object -
  Integ superclass #FIXED "relational"
  Integ inheritors #FIXED "none"
  Integ type #FIXED "relational_object"
  id ID #IMPLIED - id of object -
  objid IDREF #REQUIRED - reference id for object -
  name NAME "def_link" - name of synchro object -
  memo NMTOKENS #IMPLIED - memo attached with synchro -
  source NMTOKENS #REQUIRED - source anchor of synchro -
  sink NMTOKENS #REQUIRED - destination anchor of synchro -
  type (elementary | independent | elementary - type of synchronization
    interleaved | cyclic | chained | between objects -
    interruptible | conditional)
  displink (yes | no) no - display the synchro -
  scopetrk (intra | inter) intra - scope of synchro over track -
  scopescr (intra | inter) intra - scope of synchro over score -

- include relational superclass attributes -
Appendix B

HyTime Listing of Examples

The HyTime engine parses and interprets the documents that represent an IMMA. It understands the constructs in the document and helps construct schedules that contain multimedia objects. These schedules are executed when the application is executed at run-time. Thus, the structure of the application can be represented in a linear manner in the form of a document instance as shown below in the three examples. The run-time execution of the application depends on the end-user’s actions and thus can’t be represented in a linear manner. In many computer systems, the applications are such that their structure can be represented linearly as a document. Our mapping representation is well suited for such systems. In cases where it becomes necessary to represent very complex synchronizing situations one would then have to necessarily produce schedules for every track within the score. The following IMMA examples are from the class of applications that can be represented in a linear manner. This means that for every score there would be a schedule produced by the representation builder that will then be executed when the application is executed at run-time.

This section encodes the following three IMMA examples in HyTime.

(i) Simple Slideshow with Audio,

(ii) Slideshow with User Control, and
APPENDIX B. HYTIME LISTING OF EXAMPLES

Figure B.1: Integrator Screen Showing Slideshow with Audio

(iii) Demo Program

B.1 Slideshow with Audio

This is an example of a multimedia document encoded in the SGML/HyTime document structuring language. We call this document "ssdoc" (representing the slideshow document); it formalizes the following application. A simple slideshow consists of three slides (still images) and accompanying audio. The three slides are displayed for 10 seconds, 15 seconds, and 5 seconds, and the duration of the audio lasts through the display of the three slides.

Pseudocode:

```c
start(); /* start slideshow program */
audiostart(audio1); /* start audio */
cls(black); /* clear screen to black */
display(title1); /* display title screen */
wait(5); /* wait five seconds */
```
**APPENDIX B. HYTIME LISTING OF EXAMPLES**

```plaintext

/* display first image */
display(stillim1);

/* wait ten seconds */
wait(10);

dotrans(trans); /* perform transition */

/* display second image */
display(stillim2);

/* wait fifteen seconds */
wait(15);

dotrans(trans); /* perform transition */

/* display third image */
display(stillim3);

/* wait five seconds */
wait(5);

dotrans(trans); /* perform transition */

/* end audio */
audiostop(audio1);

/* clear screen to black */
cls(black);

/* exit slideshow program */
exit();
```

Figure B.1 shows the front-end interface of the Integrator describing the simple slideshow application in which three slides have been aligned sequentially on "window 1" and the audio has been placed on the audio track, running in parallel along with the three slides.

A conceptual representation of the slideshow example using HyTime constructs is shown in Figure B.2. The values of attributes of the components in the slideshow are defined in some Source FCS and then translated to some Target FCS. Figure B.2 gives a pictorial view of the translation of the values of the attributes. The two events schedules are governed by the two batons which project the VTUs in the Source FCS, to RTUs in the Target FCS, and in that process synchronize the audio object and the display of the slides. The class of documents must be defined before an instance of it is created. A listing of an SGML DTD for our hypothetical multimedia document, "ssdoc", is shown in section B.4. Having declared the ssdoc DTD, we can then list a "ssdoc" document instance.

**B.1.1 User View of the Application**

In this application, the user is displayed the title slide after the screen is cleared to black. The title slide is displayed for 5 seconds. After this the three still images are displayed one after another. An audio is synchronized with the display of the three slides. Special effects or transitions are performed to display the subsequent slide. The application ends with the...
end of the audio and the screen is once again cleared to black.

**B.1.2 HyTime Representation**

The document instance for the class of documents is shown below.

```xml
<!DOCTYPE ssdoc SYSTEM "/usr/arun/ssdoc.dtd" [
<!- -declare the multimedia objects used in this application- - >
<!ENTITY title1 SYSTEM "/usr/arun/stills/title1.c16" CDATA DVI >
<!ENTITY stillim1 SYSTEM "/usr/arun/stills/stillim1.c16" CDATA DVI >
<!ENTITY stillim2 SYSTEM "/usr/arun/stills/stillim2.c16" CDATA DVI >
<!ENTITY stillim3 SYSTEM "/usr/arun/stills/stillim3.c16" CDATA DVI >
<!ENTITY audio1 SYSTEM "/usr/arun/audio/audio1.avs" CDATA DVIAUD >
<!ENTITY stltrans SYSTEM "/usr/bin/trans.c" NDATA EFFECT >
<!ENTITY clear SYSTEM "/usr/bin/vds.c" NDATA VCLS >

<!- -declare the programs required to execute the multimedia objects- - >
```
APPENDIX B. HYTIME LISTING OF EXAMPLES

<!NOTATION DVI SYSTEM "/usr/bin/vshow.exe">
<!NOTATION DVIAUD SYSTEM "/usr/bin/vaudplay.exe">
<!NOTATION VCLS SYSTEM "/usr/bin/vcls.exe">
<!NOTATION EFFECT SYSTEM "/usr/bin/ trans.exe">

<!ssdoc>
<!- -the tag identifier for the start of the document instance- - >
<!ssfcs><!- -fcs with one axis: time- ->
<!- -the tag identifier for defining the FCS- - >
<!sched><!- -evsched with basegran of 1/10 second- ->
<!- -the tag identifier for defining the schedule for the single score- - >
<!- -the FCS contains two schedules: one for video - “visched” and the other for audio - “audsched” - ->
<!visched><!- -evsched with basegran of 1/10 second- ->
<!- -declare the visual objects in the schedule- - >
<!visevent id=“beginshow” exspec=“de”><visobj src=“clear”></visevent>
<!visevent exspec=“dt”><visobj src=“title1”></visevent>
<!visevent exspec=“still1”><visobj src=“stillim1”></visevent>
<!visevent exspec=“tr”><visobj src=“stltrans”></visevent>
<!visevent exspec=“still2”><visobj src=“stillim2”></visevent>
<!visevent exspec=“tr”><visobj src=“stltrans”></visevent>
<!visevent exspec=“still3”><visobj src=“stillim3”></visevent>
<!visevent exspec=“tr”><visobj src=“stltrans”></visevent>
<!visevent id=“endshow” exspec=“de”><visobj src=“clear”></visevent>

<!/visched>
<!audsched><!- -evsched with basegran of 1/10 second- ->
<!- -declare the audio object in the schedule with source as “audio1” and extent specification id “daud”- - >
<!aadevent exspec=“daud”><audobj src=“audio1”></aadevent>

<!/audsched>
<!/sched>
<!/ssfcs>

<!- -the following dimspec statements specify the dimensions for the objects used in the IMMA - ->
<!dimspec id=“dcclear”><marker1><marker1><marker2><marker5></dimspec>
<!- -the object with id “dcclear” has starting marker value 1 and ending marker value 5. This VTU will be converted to RTU by multiplying the extent by the base granule i.e. 1/10. So this object will be displayed for 5/10th seconds.- - >
<!dimspec id=“dtrans”><marker1><marker1><marker2><marker50></dimspec>
<!dimspec id=“still1”><marker1><marker1><marker2><marker100></dimspec>
APPENDIX B. HYTIME LISTING OF EXAMPLES

B.2 Slideshow with User Control

Display all still images from a source directory (for e.g., hard disk, CD-ROM, etc.) in a sequential manner. The display of the images is controlled by the selection of a mouse button - the right mouse button causes display of the next image, the left mouse button leads to display of the previous image and the application exits when both mouse buttons are selected simultaneously. An audio clip is played in the background with the presentation of the still images.

Pseudocode:

```
cls(black); /* Clear screen to black */
var done=FALSE; var count=0; var maxfiles=0; /* Declare all variables */
audio=start(audio2); /* Start audio */
display(title2); /* Display the title screen */
wait(5); /* Wait for 5 seconds */
while (findfile(".c16")) /* While there are images in the directory */
    maxfiles++; /* Count number of images */
while (!done) {
    wait until mouseclick;
    if (mouseclick == LEFT && count == 1) /* Display first image */
        display(im(1));
    else display(im(count-1)); /* Display previous image */
    if (mouseclick == RIGHT && count == maxfiles) /* Display last image */
        display(im(maxfiles));
```
else display(im(count+1));         /* Display next image */
if (mouseclick == (LEFT & & RIGHT)) {
    done=TRUE;                      /* Set flag */
    cls(black);                     /* Clear screen to black */
    exit();                         /* Exit the application */
}

Figure B.3 shows the front-end interface of the Integrator defining the Slideshow with User Control example. In this the still image object "title2" is tied to the audio object "audio2" (represented by a solid line). The dotted lines represent the scope of the control of the "ifthel" object.

### B.2.1 User View of the Application

In this application, a title slide is displayed after the screen is cleared to black. An audio is started along with the start of the display of the title slide. The first slide is displayed
and the application then waits for the end-user to interact with it. If the user selects the left mouse button, the next image in the directory is displayed. If the user selects the right mouse button, the application displays the previous image in the directory. If the user selects both mouse buttons, the application quits after clearing the screen to black.

### B.2.2 HyTime Representation

The document instance for the class of documents is shown below.

```xml
<!DOCTYPE ssucdoc SYSTEM "/usr/arun/ssucloc.dtd">
<!ENTITY title2 SYSTEM "/usr/arun/stills/title2.cl6" CDATA DVI >
<!ENTITY stillim SYSTEM "/usr/arun/stills/*.c16" CDATA DVI >
<!ENTITY audio2 SYSTEMM "/usr/arun/audio/audio2.avs" CDATA DVIAUD >

<ssucdoc>
  <ssucfcs>
    <visevent id:.;:: "beginshow" exspee:.;:: "de" > <visobj sre:.;:: "clear" > < /visevent>
    <integevent exspec:.;:: "dvar" > <var script:.;:: var1 > done </var > </integevent>
    <integevent exspec:.;:: "dwhile" > <while script:.;:: while1 > < /while > </integevent>
    <integevent exspec:.;:: "dwait" > <wait script:.;:: wait1 > < /wait > </integevent>
  </ssucfcs>
</ssucdoc>
```

The `while` event has a condition part and an action part defined as follows:

```xml
<cond>(!done) </cond>
```
APPENDIX B. HYTIME LISTING OF EXAMPLES

<action>
   <integevent exspec="dmouse" mouse script="mouse1"></mouse>
</integevent>

<integevent exspec="dif">
   <ifthen>
      <cond> (mousedclick == LEFT) && (count == 1) </cond>
      <then> (display (im(1))) </then>
      <else> (display (im(count-1))) </else>
   </ifthen>
</integevent>

<integevent exspec="dif">
   <ifthen>
      <cond> (mousedclick == RIGHT) && (count == maxfiles) </cond>
      <then> (display (im(maxfiles))) </then>
      <else> (display (im(count+1))) </else>
   </ifthen>
</integevent>

<integevent exspec="dif">
   <ifthen>
      <cond> (mousedclick == (LEFT || RIGHT)) </cond>
      <then> (done = TRUE;) (cls(black);) (exit;) </then>
   </ifthen>
</integevent>

</action>

<while>
   <integevent>
      <visevent id="endshow" exspec="dc"><visobj src="clear"></visevent>
   </integevent>
</while>

<!- -The following dimspec statements specify the dimension specifications for the objects used in the IMMA- ->
<dimspec id="dc" <marker1><marker>1 <marker2><marker>5>
<dimspec id="dvar" <marker2><marker>-1>
<dimspec id="da">
   <marker1><marker><dimref first>begshow</dimref>
   <marker2><marker><dimref last>endshow</dimref>
<dimspec id="dt" <marker1><marker>1 <marker2><marker>50>
<dimspec id="dwait" <marker2><marker>-1>
<dimspec id="dwhile" <marker2><marker>-1>
APPENDIX B. HYTIME LISTING OF EXAMPLES

```xml
<dimspec id="dmouse"/><marker2><marker>-l
<dimspec id="dif"> <marker2><marker>-1
<!- -the following declarations specify statements to be evaluated by some controlling program (ccode)- ->
<script id=var1 notation=ccode>var done=FALSE; </script>
<!- -the external controlling program that controls the statement in the script identified by "var1" is a ccode- ->
<script id=var2 notation=ccode>var count=0; </script>
<script id=var3 notation=ccode>var maxfiles=0; </script>
<script id=wait1 notation=ccode>wait(argl); </script>
<script id=while1 notation=ccode>
while (findfile(*.cl6})
maxfiles++;
</script>
<script id=mousel notation=ccode>getmousedclick; </script>
</ssucdoc>
```

B.3 Demo Program

This scenario shows how applications using menu can be built using the Integrator. This is a demo program in which the user can view images in a slide show, play a video sequence, view a graphics segment, or take part in a quiz. The menu consists of four stills where the user can click in any one of the sections to view that particular section. This program uses multiple scores through the use of composite objects.

Pseudocode:

```plaintext
cls(black); /* Clear screen to black */
audiostream(audio3); /* Start audio */
display(title3); /* Display the title screen */
wait(5); /* Wait for 5 seconds */
repeat {
   initialize();
   if ((mousedclick == LEFT) & (areaselected == S1)) {
      videostart(video); /* Play video sequence */
      return(); /* Go back to start of loop */
   }
   if ((mousedclick == LEFT) && (areaselected == S2)) {
      slideshow(); /* Call slide show program */
      return(); /* Go back to start of loop */
   }
```

120
APPENDIX B. HYTIME LISTING OF EXAMPLES

Figure B.4: Integrator Screen Showing Demo Program

Figure B.5: Integrator Screen Showing Slideshow Composite
APPENDIX B. HYTIME LISTING OF EXAMPLES

Figure B.6: Integrator Screen Showing Quiz Composite

```c
)
if ((mouseclick == LEFT) && (areaselected == S3)) {
    graphics(); /* Call the graphics routine */
    return(); /* Go back to start of loop */
}
if ((mouseclick == LEFT) && (areaselected == S4)) {
    quiz(); /* Call quiz program */
    return(); /* Go back to start of loop */
}
) until (mouseclick == RIGHT);
cls(black); /* Clear screen to black */
exit(); /* Exit the application */
initialize() {
    display(S1); display(S2); /* Display the four images */
    display(S3); display(S4);
    text("Left mouseclick on image to explore further");
    text("Right mouseclick to quit program");
    wait for mouseclick;
}
```
APPENDIX B. HYTIME LISTING OF EXAMPLES

slideshow() {
    cls(black); /* Clear screen to black */
    var done=FALSE; count=0; maxfiles=0; /* Declare all variables */
    audiosketch(audio2); /* Start audio */
    display(title2); /* Display the title screen */
    wait(5); /* Wait for 5 seconds */
    while (findfile("*c16") /* While there are images in the directory */
        maxfiles++;
    count=0;
    while (!done) {
        wait until mouseclick;
        if (mouseclick == LEFT & count == 1)
            display(im(1)); /* Display first image */
        else display(im(count-1)); /* Display previous image */
        if (mouseclick == RIGHT & count == maxfiles)
            display(im(maxfiles)); /* Display last image */
        else display(im(count+1)); /* Display next image */
        if (mouseclick == (LEFT & RIGHT)) {
            done=TRUE; /* Set flag */
            cls(black); /* Clear screen to black */
        }
        exit(); /* Exit the application */
    }
}
}

quiz() {
    var count=0;
    text("Question 1"); /* Ask the first question */
    wait for input; /* Wait for an input */
    if ("input == Answer1") { /* Evaluate the answer */
        text("Correct");
        count++;
    }
    else {
        text("Wrong");
        count--; /* Clear screen to black */
    }
    cls(black); /* Ask the second question */
    text("Question 2");
    wait for input; /* Wait for an input */
APPENDIX B. HYTIME LISTING OF EXAMPLES

```c
if ("input == Answer2") { /* Evaluate the answer */
    text("Correct");
    count++;
}
else {
    text("Wrong");
    count--; 
}
cls(black);  /* Clear screen to black */
text("Question 3");  /* Ask the third question */
if ("input == Answer 3") { /* Evaluate the third answer */
    text("Correct");
    count++;
}
else {
    text("Wrong");
    count--; 
}
cls(black);  /* Clear the screen to black */
if (count <2)
    text("Better luck next time");
else text ("Good! Well done");
text("Your score is, count");  /* Display the score obtained */
wait(5);  /* Wait for 5 seconds */
cls(black);  /* Clear screen to black */
exit();  /* Quit this program */
```

B.3.1 User View of the Application

In this application, the title slide is displayed after the screen is cleared to black. The title slide is displayed for 5 seconds. Next, a menu consisting of four quarter screen images is displayed and a text message appears on the screen which instructs the user to select any one of the choices. Depending on the end-user’s selection, the application executes a video sequence, a slideshow program, a graphics sequence, or a quiz program. The application displays the menu once again when the selected menu item is finished executing. If the
APPENDIX B. HYTIME LISTING OF EXAMPLES

user clicks the right mouse button, the application quits the program clearing the screen to black.

B.3.2 HyTime Representation

The document instance for the class of documents is shown below.

```
<!DOCTYPE demodoc SYSTEM "/usr/arun/demodoc.dtd" [ 
<!ENTITY title3 SYSTEM "/usr/arun/stills/title3.c16" CDATA DVI > 
<!ENTITY title2 SYSTEM "/usr/arun/stills/title2.c16" CDATA DVI > 
<!ENTITY s1 SYSTEM "/usr/arun/stills/s1.c16" CDATA DVI > 
<!ENTITY s2 SYSTEM "/usr/arun/stills/s2.c16" CDATA DVI > 
<!ENTITY s3 SYSTEM "/usr/arun/stills/s3.c16" CDATA DVI > 
<!ENTITY s4 SYSTEM "/usr/arun/stills/s4.c16" CDATA DVI > 
<!ENTITY stillim SYSTEM "/usr/arun/stills/*.c16" CDATA DVI > 
<!ENTITY audio2 SYSTEM "/usr/arun/audio/audio2.avs" CDATA DVIAUD > 
<!ENTITY video SYSTEM "/usr/arun/audio/video.avs" CDATA DVIVID > 
<!ENTITY graphics SYSTEM "/usr/arun/audio/graphics.avs" CDATA DVIGR > 

<!NOTATION DVI SYSTEM "/usr/bin/vshow.exe" > 
<!NOTATION DVIAUD SYSTEM "/usr/bin/vaudplay.exe" > 
<!NOTATION DVIVID SYSTEM "/usr/bin/vvidplay.exe" > 
<!NOTATION DVIGR SYSTEM "/usr/bin/vgrplay.exe" > 

<demodoc>
  <demofcs><!- -fcs with one axis: time- - >
    <sched><!- -evschced with basegran of 1/10 second- - >
      <integevent id="beginpgm" exspec="dcompu"> <compu script=compu1></compu>
    </integevent>
    <visevent id="title" exspec="dstill" > <visobj src="title3"></visevent>
    <audevent exspec="daud"> <audobj src="audio3"cl></audevent>
    <integevent exspec="drpt" >
      <rptuntil>
        <repeat>
          <visevent id="stllmenul" exspec="dstill"> <visobj src="s1"></visevent>
          <visevent exspec="dstill"> <visobj src="s2"></visevent>
        </repeat>
      </rptuntil>
    </integevent>
  </sched>
</demofcs>
```

125
APPENDIX B. HYTIME LISTING OF EXAMPLES

<visevent exspec="dstilll"> <visobj src="s3"> </visevent>

<visevent id="stlmenu4" exspec="dstilll"> <visobj src="s4"> </visevent>

<integevent exspec="dtex"> <text script=text1></text> </integevent>

<integevent exspec="dselmou"> <selmouse script=selmou1></selmouse> </integevent>

<integevent exspec="dif"> </integevent>

<integevent exspec="dif"> <ifthen> <cond>((mouseclick == LEFT) && (areaselected == S1)) </cond> <then>(videostart(videol); return()); </then> </ifthen> </integevent>

<integevent exspec="dif"> <ifthen> <cond>((mouseclick == LEFT) && (areaselected == S2)) </cond> <then> <integevent exspec="dcompo"> <compo> <!--include slideshow with user control (listed below) here--> </compo> </integevent> </then> </ifthen> </integevent>

<integevent exspec="dif"> <ifthen> <cond>((mouseclick == LEFT) && (areaselected == S3)) </cond> <then>(graphics(); return()); </then> </ifthen> </integevent>

<integevent exspec="dif"> <ifthen> <cond>((mouseclick == LEFT) && (areaselected == S4)) </cond> <then> <integevent exspec="dcompo"> <compo> <!--include quiz program (listed below) here--> </compo> </integevent> </then> </ifthen> </integevent>

126
APPENDIX B. HYTIME LISTING OF EXAMPLES

```
<integevent>
</repeat>
</until>

<integevent exspec="dmouse"> <mouse script=mousel> </mouse> </integevent>
</until>
</rptuntil>
</integevent>

<visevent id="endpgm" exspec= "dc" > <visobj src="clear"> </visevent>
</sched>
</demofcs>

<!- -the quiz program starts here. It actually will occur in place marked above- ->
<sched>

<integevent exspec="dvar"> <var script=var4>count1</var> </integevent>

<integevent id="textq1" exspec="dtext"> <text script=text2> </text> </integevent>

<integevent exspec="dinput"> <input script=input1> </input> </integevent>

<integevent exspec="dif">

<ifthen>
  <cond>(input == "Answer 1")</cond>
  <then>(printf("Correct"); (count++);) </then>
  <else>(printf("Wrong"); (count--);) </else>
</ifthen>

</integevent>

<visevent exspec= "de" > <visobj src= "clear"> </visevent>

<integevent id="textq2" exspec= "dtext"> <text script=text3> </text> </integevent>

<integevent exspec= "dinput"> <input script=input2> </input> </integevent>

<integevent exspec= "dif">

<ifthen>
  <cond>(input == "Answer 2")</cond>
  <then>(printf("Correct"); (count++);) </then>
  <else>(printf("Wrong"); (count--);) </else>
</ifthen>

</integevent>

<visevent exspec= "de" > <visobj src= "clear"> </visevent>

<integevent id="textq3" exspec= "dtext"> <text script=text4> </text> </integevent>

<integevent exspec= "dinput"> <input script=input3> </input> </integevent>

<integevent exspec= "dif">

<ifthen>
  <cond>(input == "Answer 3")</cond>
  <then>(printf("Correct"); (count++);) </then>
  <else>(printf("Wrong"); (count--);) </else>
</ifthen>

</integevent>

<visevent exspec= "de" > <visobj src= "clear"> </visevent>

<integevent id="textq4" exspec= "dtext"> <text script=text5> </text> </integevent>

<integevent exspec= "dinput"> <input script=input4> </input> </integevent>

<integevent exspec= "dif">

<ifthen>
  <cond>(input == "Answer 4")</cond>
  <then>(printf("Correct"); (count++);) </then>
  <else>(printf("Wrong"); (count--);) </else>
</ifthen>

</integevent>

<visevent exspec= "de" > <visobj src= "clear"> </visevent>

<integevent id="textq5" exspec= "dtext"> <text script=text6> </text> </integevent>

<integevent exspec= "dinput"> <input script=input5> </input> </integevent>

<integevent exspec= "dif">

<ifthen>
  <cond>(input == "Answer 5")</cond>
  <then>(printf("Correct"); (count++);) </then>
  <else>(printf("Wrong"); (count--);) </else>
</ifthen>

</integevent>

<visevent exspec= "de" > <visobj src= "clear"> </visevent>

<integevent id="textq6" exspec= "dtext"> <text script=text7> </text> </integevent>

<integevent exspec= "dinput"> <input script=input6> </input> </integevent>

<integevent exspec= "dif">

<ifthen>
  <cond>(input == "Answer 6")</cond>
  <then>(printf("Correct"); (count++);) </then>
  <else>(printf("Wrong"); (count--);) </else>
</ifthen>

</integevent>

<visevent exspec= "de" > <visobj src= "clear"> </visevent>

<integevent id="textq7" exspec= "dtext"> <text script=text8> </text> </integevent>

<integevent exspec= "dinput"> <input script=input7> </input> </integevent>

<integevent exspec= "dif">

<ifthen>
  <cond>(input == "Answer 7")</cond>
  <then>(printf("Correct"); (count++);) </then>
  <else>(printf("Wrong"); (count--);) </else>
</ifthen>

</integevent>

<visevent exspec= "de" > <visobj src= "clear"> </visevent>

<integevent id="textq8" exspec= "dtext"> <text script=text9> </text> </integevent>

<integevent exspec= "dinput"> <input script=input8> </input> </integevent>

<integevent exspec= "dif">

<ifthen>
  <cond>(input == "Answer 8")</cond>
  <then>(printf("Correct"); (count++);) </then>
  <else>(printf("Wrong"); (count--);) </else>
</ifthen>

</integevent>

<visevent exspec= "de" > <visobj src= "clear"> </visevent>

<integevent id="textq9" exspec= "dtext"> <text script=text10> </text> </integevent>

<integevent exspec= "dinput"> <input script=input9> </input> </integevent>

<integevent exspec= "dif">

<ifthen>
  <cond>(input == "Answer 9")</cond>
  <then>(printf("Correct"); (count++);) </then>
  <else>(printf("Wrong"); (count--);) </else>
</ifthen>

</integevent>

<visevent exspec= "de" > <visobj src= "clear"> </visevent>

<integevent id="textq10" exspec= "dtext"> <text script=text11> </text> </integevent>

<integevent exspec= "dinput"> <input script=input10> </input> </integevent>

<integevent exspec= "dif">

<ifthen>
  <cond>(input == "Answer 10")</cond>
  <then>(printf("Correct"); (count++);) </then>
  <else>(printf("Wrong"); (count--);) </else>
</ifthen>

</integevent>

<visevent exspec= "de" > <visobj src= "clear"> </visevent>

<integevent id="textq11" exspec= "dtext"> <text script=text12> </text> </integevent>

<integevent exspec= "dinput"> <input script=input11> </input> </integevent>

<integevent exspec= "dif">

<ifthen>
  <cond>(input == "Answer 11")</cond>
  <then>(printf("Correct"); (count++);) </then>
  <else>(printf("Wrong"); (count--);) </el
APPENDIX B. HYTIME LISTING OF EXAMPLES

```c
<then>(printf("Correct"); (count++)); </then>
<else>(printf("Wrong"); (count--)); </else>
</ifthen>
</integevent>

<visevent exspec= "dc"> <visobj src= "clear" > </visevent>
<integevent exspec= "dif" >
<ifthen>
<cond>(count <3) </cond>
<then>(printf("Better luck next time")) </then>
<else>(printf("Good. Well done") </else>
</ifthen>
</integevent>

<integevent exspec= "dwait"> <wait script=wait1>5 </wait> </integevent>

<integevent exspec= "dc"> <visobj src= "clear" > </visevent>

opn /sched>
<!- -the quiz program ends here- ->

<!- -the slideshow program starts here- ->

<sched> <!- -evsched with basegran of 1/10 second- ->
<integevent id= "beginshow" exspec= "dc" > <visobj src= "clear" > </visevent>
<integevent exspec= "dvar" > <var script=var1>done</var> </integevent>
<integevent exspec= "dvar" > <var script=var2>count</var> </integevent>
<integevent exspec= "dvar" > <var script=var3>maxfiles</var> </integevent>
<audevent exspec="da"> <audobj src="audio2"cl </audevent>
<visevent exspec= "dt" > <visobj src= "title2" > </visevent>
<integevent exspec= "dwait"> <wait script=wait1>5 </wait> </integevent/cl
<integevent exspec= "dwhile"> <while script=while1> </while> </integevent/cl
<integevent exspec= "dwhile"> <while>
<cond>(!done) </cond>
</cond>
</action>
<integevent exspec= "dmouse" > <mouse script="mouse1" > </mouse>
</integevent>
<integevent exspec= "dif" >
<ifthen>
<cond>((mouseclick == LEFT) && (count == 1))
</cond>
<then>(display (im(1))) </then>
<else>(display (im(count-1))) </else>
</ifthen>
</integevent>
```

128
APPENDIX B. HYTIME LISTING OF EXAMPLES

```xml
<integevent>
  <integevent exspec="dif">
    <ifthen>
      <cond>((mousedick == RIGHT) && (count == maxfiles))
      <then>(display (im(maxfiles)))</then>
      <else>(display (im(count+1)))</else>
    </ifthen>
  </integevent>
  <integevent exspec="dif">
    <cond>((mousedick == (LEFT && RIGHT))</cond>
    <then>(((done = TRUE) (cls(black)); (exit;)))</then>
  </ifthen>
</integevent>

</integevent>

</action>
</integevent>

<visevent id="endshow" exspec="de"> <visobj src="clear"></visevent>

</sched>
<!- -the slideshow program ends here- ->

<!- The following dimspect statements specify the dimension specifications for the objects used in the IMMA- ->

<dimspec id="dcompu"> <marker1><marker>1 <marker2><marker>-1</marker2></marker1></marker1> <marker2><marker>-1</marker2></marker2>
<dimspec id="dstill" > <marker1><marker>1 <marker2><marker>-1</marker2></marker1></marker1> <marker2><marker>-1</marker2></marker2>
<dimspec id="daud" > <marker1><marker>-1 <marker2><marker>-1</marker2></marker1></marker1> <marker2><marker>-1</marker2></marker2>
<dimspec id="dtext" > <marker1><marker>-1 <marker2><marker>-1</marker2></marker1></marker1> <marker2><marker>-1</marker2></marker2>
<dimspec id="dif" > <marker1><marker>-1 <marker2><marker>-1</marker2></marker1></marker1> <marker2><marker>-1</marker2></marker2>
<dimspec id="dvar" > <marker1><marker>-1 <marker2><marker>-1</marker2></marker1></marker1> <marker2><marker>-1</marker2></marker2>
<dimspec id="dinput" > <marker1><marker>-1 <marker2><marker>-1</marker2></marker1></marker1> <marker2><marker>-1</marker2></marker2>
```

129
APPENDIX B. HYTIME LISTING OF EXAMPLES

<!- - include dimspec statements from Scenario 2 - ->
<!- - The following declarations specify statements to be evaluated by some controlling program - ->
<script id=compu1 notation=ccode>cls(black); </script>
<script id=text1 notation=ccode>"Select one" </script>
<script id=selmoul notation=ccode>
  areaselected = getarea();
</script>
<script id=text2 notation=ccode>- -Display Question 1- - </script>
<script id=text3 notation=ccode>- -Display Question 2- - </script>
<script id=text4 notation=ccode>- -Display Question 3- - </script>
<script id=var2 notation=ccode>var count2 = 0; </script>
<script id=input1 notation=ccode>input = getchar(); </script>
<!- - include script statements from Scenario 2 - ->
<script id=var1 notation=ccode>var done=FALSE; </script>
<script id=var2 notation=ccode>var count=0; </script>
<script id=var3 notation=ccode>var maxfiles=0; </script>
<script id=waitl notation= ccode>wait(arg1); </script>
<script id=whilel notation=ccode>
  while (findfile("*.c16")
      maxfiles++;
</script>
<script id=mousel notation= ccode>getmousedclick; </script>

B.4 DTD for the IMMA Examples

This section lists the DTD that specifies the architectural forms and the constructs used to define the IMMA examples shown above.

<!- - DTD for DOCTYPEs ssdoc, ssucdoc, and demodoc - ->
<!ENTITY % resmeas - -resource element types: measurement module- -
APPENDIX B. HYTIME LISTING OF EXAMPLES

"dimspec | dimref" >
<!ENTITY % object - -element type allowed in events usable integrator objects- -
"audio | video | ... | hyper | synchro" - -include names of other objects- - >
<!ENTITY % fcs - -finite coordinate space element type- -
"ssfcs | ssucfcs | demofcs" >

<!ELEMENT (ssdoc | ssucdoc | demodoc) - -top level document element- -
    -O (
        (%fcs; | %resmeas; | %object;)+ 
    ) >
<!ATTLIST (ssdoc | ssucdoc | demodoc)
    ssucdoc CDATA #FIXED "ssucdoc"
    demodoc CDATA #FIXED "demodoc"
    hyqcnt NUMBER #FIXED 32 - -Default: 4,294,967,295- -
    docmdu CDATA #FIXED "virtime 1 1" - -Default: set by FCS or axis- -
>
<!ELEMENT (ssfcs | ssucfcs | demofcs) - - (sched)+ >
<!ATTLIST (ssfcs | ssucfcs | demofcs)
    id ID #IMPLIED
    fcsmdu CDATA #FIXED "" - -Default: equal to docmdu- -
    axisdefs CDATA #FIXED "dtime"
>
<!ELEMENT dtime - -axis definition for a HyTime FCS- -
- -It extends the FCS definition- -
APPENDIX B. HYTIME LISTING OF EXAMPLES

<!ATTLIST dtime
  NAME #FIXED axis
  axismeas
  cdatta #FIXED " " #Default: #FIXED in DTD -
  axismdu
  cdatta #FIXED " " #Default: equal to fcsmdu-
  axismun
  cdatta #FIXED " " #Default: #FIXED in DTD-

<!ELEMENT dimspec
  hytime
  cdatta #FIXED "dimspec xdimspec"
  id id #IMPLIED
  overrun (error | trim) error
  range
  (axis | previous | next) #CURRENT
  dimsrc
  cdatta #FIXED " " #Default: equal to fcsmdu-
  NAMES #CONREF
  selcomp
  (first | last | qcount) qcount

<!ELEMENT markerl
  hytime
  cdatta #FIXED #PCDATA>
<!ELEMENT marker2
  hytime
  cdatta #FIXED #PCDATA>
<!ELEMENT marker
  hytime
  cdatta #FIXED #PCDATA>
APPENDIX B. HYTIME LISTING OF EXAMPLES

<!ATTLIST marker HyTime CDATA #FIXED marker >

<!ELEMENT dimref - Dimension reference- -
  O (#PCDATA) >
<!ATTLIST dimref HyTime NAME #FIXED dimref
id ID #IMPLIED
  - -dimref attributes- -
  lextypedata CDATA #FIXED "#CONTENT IDREF axisref GI"
axisref - -Axis referenced- -
CDATA " " - -Default: one axis- -
selcomp - -Selected component of dimension- -
  (first | last | qcnt) qcnt
flip - -Last is normally counted from end of range (negative),
  and first from start of range (positive). Qcnt is also
  normally positive. Flip counts last from start of range
  (making it positive), and first from end of range (making it
  negative). Flip also makes qcnt negative. -
  (flip | noflip) noflip >

<!ELEMENT sched - -
  ((visched | audsched | integevent)+) >
<!ATTLIST sched HyTime NAME #FIXED evsched
id ID #IMPLIED
  - -sched use: evsched baton wand- -
axisord - -order of axis in schedule- -
CDATA #FIXED "dtime" - -Default: #FIXED in DTD- -
sorted - -representation of elements is sorted by order of position
  on first axis of schedule- -
CDATA #FIXED "unsorted" - -Default: #FIXED in DTD- -
apporder - -order of schedule elements is significant to the application
  and must be preserved- -
CDATA #FIXED "order" - -Default: #FIXED in DTD- -
basegran - -Base granule for each axis- -
CDATA #FIXED "1 10" - -Default: #FIXED in DTD- -
>
APPENDIX B. HYTIME LISTING OF EXAMPLES

<!ELEMENT integevent - event used in the document -
- O (%object; )>
<!ATTLIST integevent
    id ID #IMPLIED
    exspec CDATA #REQUIRED
    basegran CDATA #REQUIRED
    NAMES #FIXED "1 1"
>
<!ELEMENT visched - schedule of visual events -
- (visevent+) >
<!ATTLIST visched
    HyTime NAME #FIXED evsched
    id ID #IMPLIED
    axisord CDATA #FIXED "dtime" - Default: #FIXED in DTD -
    sorted CDATA #FIXED "unsorted" - Default: #FIXED in DTD -
    apporder CDATA #FIXED "order" - Default: #FIXED in DTD -
    basegran CDATA #FIXED "1 10" - Default: #FIXED in DTD -
>
<!ELEMENT visevent - visual events -
- O (visobj; )>
<!ATTLIST visevent
    id ID #IMPLIED
    exspec CDATA #REQUIRED
    basegran CDATA #REQUIRED

134
APPENDIX B. HYTIME LISTING OF EXAMPLES

<!ELEMENT audsched (
  <!ATTLIST audsched
    HyTime NAME #FIXED evsched
    id ID #IMPLIED
    axisord CDATA #FIXED "dtime" - Default: #FIXED in DTD -
    sorted CDATA #FIXED "unsorted" - Default: #FIXED in DTD -
    apporder CDATA #FIXED "order" - Default: #FIXED in DTD -
    basegran CDATA #FIXED "1 10" - Default: #FIXED in DTD -
    )
  )

<!ELEMENT audevent (audobj)
<!ATTLIST audevent
    id ID #IMPLIED
    exspec CDATA #REQUIRED
    basegran CDATA #-base granule for each axis-
  )

<!ELEMENT visobj (#PCDATA)>

135
APPENDIX B. HYTIME LISTING OF EXAMPLES

The following tables, Table B.1 and Table B.2, list the source and the purpose of the different elements and keywords used in the three IMMA examples and the DTD that the examples conform to.
APPENDIX B. HYTIME LISTING OF EXAMPLES

Table B.1: Elements and Keywords Used in the Examples and DTD

<table>
<thead>
<tr>
<th>Elements and Keywords</th>
<th>Source</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Integr</td>
<td>HyTime</td>
</tr>
<tr>
<td>audevent</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>audobj</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>audsched</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>baton</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>ccdata</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>CData</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>compo</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>demodoc</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>demofcs</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>dimspec</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>dimref</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>DOCTYPE</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>dttime</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DTD</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>DVIGR</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>DVIVID</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>DVI</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>DVIAUD</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>EFFECT</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>ENTITY</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>exspec</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>fcs</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>first</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>id</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>ifthe</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>integevent</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>input</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>last</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>link</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>marker</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>
### APPENDIX B. HYTIME LISTING OF EXAMPLES

Table B.2: Elements and Keywords Used in the Examples and GTD (contd.)

<table>
<thead>
<tr>
<th>Elements and Keywords</th>
<th>Source</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integ Obj.</td>
<td>HyTime</td>
<td>MHEG</td>
</tr>
<tr>
<td>marker1</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>marker2</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>NDATA</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>notation</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>NOTATION</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>object</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>resmeans</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>sched</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>script</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>selmouse</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>src</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ssdoc</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ssfcs</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>ssudoc</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>ssucfcs</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>SUPPORT</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>SYSTEM</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>text</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>var</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>VCLS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>visched</td>
<td></td>
<td></td>
</tr>
<tr>
<td>visevent</td>
<td></td>
<td></td>
</tr>
<tr>
<td>visobj</td>
<td></td>
<td></td>
</tr>
<tr>
<td>wait</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>while</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>
Appendix C

Integrator Meta-DTD

The following listing is the SGML Declaration and the Integrator meta-DTD that has to be used to define any IMMA developed using the Integrator. The applications built using the Integrator can be encoded in HyTime and would then conform to the rules defined by the meta-DTD.

```xml
<!SGML "ISO 8879-1986" - - SGML Declaration for DOCTYPE Integrator - -
CHARSET BASESET "ISO 646-1983//CHARSET
International Reference Version (IRV)//ESC 2/5 4/0"
DESCSET 0 9 UNUSED
  9 2 9
 11 2 UNUSED
 13 1 13
 14 18 UNUSED
 32 95 32
127 1 UNUSED
CAPACITY PUBLIC "ISO 8879-1986//CAPACITY Reference // EN"
SCOPE DOCUMENT
SYNTAX SHUNCHAR CONTROL
  0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
 19 20 21 22 23 24 25 26 27 28 29 30 31 127 255
BASESET "ISO 646-1983//CHARSET
International Reference Version (IRV)//ESC 2/5 4/0"
DESCSET 0 128 0
FUNCTION RE 13
```
APPENDIX C. INTEGRATOR META-DTD

RS . 10
SPACE 32
TAB SEPCHAR 9

NAMING
LCNMSTRT " "
UCNMSTRT " "
LCNMCHAR "." 
UCNMCHAR "." 
NAMECASE GENERAL NO ENTITY NO

DELIM GENERAL SGMLREF
SHORTREF NONE

NAMES SGMLREF
QUANTITY SGMLREF

FEATURES MINIMIZE DATATAG NO OMMITTAG NO RANK NO SHORTTAG NO
LINK SIMPLE NO EXPLICIT NO IMPLICIT NO
OTHER CONCUR NO SUBDOC NO FORMAL NO

APPINFO NONE

> 

<!DOCTYPE INTEGDOC PUBLIC "-//VPI&SU//DTD SGML: Integrator Document//EN" "Integ.Rules"
<!ENTITY % hytime PUBLIC "-//ANSI X3V1.8M//DTD Hypermedia/Time-based Document//EN" "hytime.dtd" >

- -include objects and attributes listing here- -
- -include HyTime listing here- -

%hytime;
] >
- -end of DTD for doctype Integrator- -
Appendix D

Glossary

Following listing includes terms and definitions that are mentioned in this thesis report. These definitions are intended to be a compact description rather than an exhaustive list. For a detailed description please refer to the references on the Integrator, SGML and HyTime standards.

architectural form set of rules that the application designers can apply in their document type definitions.

ATTLIST attribute definition list for an element.

attribute qualifier indicating a certain property of a given element, other than its type [62].

audsched schedule of audio events.

audobj audio object.

basegran base granule for an axis.

baton schedule of projector scopes that govern the projection of event schedules affected by the baton [52].
APPENDIX D. GLOSSARY

CDATA  character data - elements may contain only SGML characters.

core event “ce”, an occurrence of an object. Its extent specification specifies the position and extent of the object [52].

core event sequence “ces”, sequence of core events.

dimref  dimension reference.

dimspec  dimension specification.

document  collection of textual information possibly augmented with graphics, tabular, and numeric data [62].

DTD  Document Type Declaration.

element  part of a document which is a logical entity [62].

ENTITY  unit of information that may be referred to by a symbol in a DTD or a document instance [62].

event  see core event.

evsched  event schedule.

event sequence “evseq”, event sequence.

FCS  Finite Coordinate Space. This defines a set of coordinate axes and provides for the scheduling of objects.

FIXED  defines the attribute to have the same value always.

HyTime  Hypermedia/Time-Based Structuring Language.

HyTime engine program that recognizes HyTime constructs in documents and performs application-independent processing of them [52].
**APPENDIX D. GLOSSARY**

id attribute to identify some object.

idref attribute to refer to some object by specifying the id value.

**IMMA** Interactive MultiMedia Application.

**IMPLIED** attribute value need not be given; the application chooses a default.

**integevent** integrator event.

**marker** axis marker

**marker1** first marker specification of a dimension specification.

**marker2** second marker specification of a dimension specification.

**MHEG** Multimedia Hypermedia Experts Group.

**mnuseq** multimedia sequence; sequence containing multimedia objects.

**NDATA** non-SGML data; data consists of other than valid SGML characters.

**NOTATION** a way of indicating parts of a document which follow markup rules other than those defined by SGML, such as TeX, Encapsulated PostScript, etc. [62].

**object** architectural forms allowed in events.

**orelation** attribute to specify relationship between objects.

**PCDATA** parsed character data; data characters that occur in marked up text that are not recognized as markup during parsing.

**Precedence Rules** rules that provide a framework to apply a consistent design when building an IMMA.

**representation builder** program that builds the intermediate representation of an IMMA.

**REQUIRED** means that the attribute must always be given a value in markup [62].
APPENDIX D. GLOSSARY

resmeas resource architectural form (measurement module in HyTime).

rtu real time unit; time in every-day sense, as measured in seconds, minutes, hours, etc.

sce special core event; used to formalize the composite construct.

sched schedule.

score working area in the front end interface of the Integrator (based on the musical score notation).

script element to define an external controlling program to execute parts of a document not conforming to SGML.

SDATA specific character data; data consists of valid SGML characters that is not analyzed by the parser [62].

SGML Structured Generalized Markup Language.

src source; attribute to identify the source of an object.

supportive objects in the Integrator that support the development of an IMMA but cannot be used in the IMMA.

timeline top level time controller which is controlled by the system clock.

track defines a temporal sequence of multimedia objects placed on it; a logical device is associated with each track.

usable objects in the Integrator that can be used in defining an IMMA.

visched visual schedule; schedule consisting of visual objects such as still images, video, etc.

vtu virtual time unit.
VITA

Arun Narasimhan was born in Madras, India, on the 16th of May, 1967. He obtained his Bachelor of Engineering degree in Computer Science and Engineering from the College of Engineering, Guindy, Anna University, Madras, India, in May 1989. He then enrolled in the Computer Science Master’s program at Virginia Polytechnic Institute and State University in the Fall of 1989. Upon completion of his Master's, he intends to work for Western Geophysical division of Western Atlas International in Houston, Texas.

Arun Narasimhan