
**Information technology — Computer
graphics and image processing —
Presentation Environment for Multimedia
Objects (PREMO) —**

Part 4:

**Modelling, rendering and interaction
component**

*Technologies de l'information — Infographie et traitement d'images —
Environnement de présentation d'objets multimédia (PREMO) —*

Partie 4: Composant pour la modélisation, le rendu et l'interaction



Contents

Foreword.....vii

Introduction viii

1 Scope 9

2 Normative references.....9

3 Definitions.....10

3.1 PREMO Part 1 definitions 10

3.2 PREMO Part 2 definitions 10

3.3 PREMO Part 3 definitions 10

3.4 Additional Definitions 10

4 Symbols and abbreviations 12

5 Conformance 12

6 Overview of the Modelling, Rendering and Interaction Component. 12

6.1 Introduction 12

6.2 Overview 12

6.3 Devices for Modelling, Rendering, and Interaction..... 16

6.4 Primitives and Coordinates..... 17

6.4.1 Introduction..... 17

6.4.2 Coordinates 17

6.4.3 The Primitive Hierarchy in PREMO 17

 6.4.3.1 Overview 17

 6.4.3.2 Captured Primitives 18

 6.4.3.3 Form Primitives 18

 6.4.3.4 Modifier Primitives 18

 6.4.3.5 Reference Primitives 19

 6.4.3.6 Structured Primitives 19

 6.4.3.7 Tracer Primitives 19

 6.4.3.8 Wrapper Primitives 19

6.4.4 Primitives and MRI Devices 19

6.5 Scene 20

6.6 Interaction 21

6.7 Coordinators 21

6.8 Dependencies on other Parts 23

6.9 Subtyping Diagram 23

7 Coordinates 23

 7.1 General Coordinates 23

 7.2 Colour 24

 7.3 TimeLocation 25

8 Primitives 25

 8.1 Introduction 25

 8.2 Captured Primitives 27

 8.3 Form Primitives 27

 8.3.1 Introduction 27

 8.3.2 Audio Primitives 27

 8.3.3 Geometric Primitives 28

 8.3.4 Tactile Primitives 28

 8.3.5 Text Primitives 28

 8.4 Modifier Primitives 28

 8.4.1 Introduction 28

 8.4.2 Acoustic Modifiers 29

 8.4.3 Structural Modifiers 29

 8.4.4 TimeFrame Modifiers 29

 8.4.5 Visual Modifiers 30

 8.5 Reference Primitives 30

 8.5.1 References 30

 8.5.2 The Name Object Type 30

 8.6 Structured Primitives 31

 8.6.1 Introduction 31

 8.6.2 Aggregate 31

 8.6.3 TimeComposite 32

 8.7 Tracer Primitives 35

 8.8 Wrapper Primitives 36

9 Modelling, Rendering and Interaction Device 36

 9.1 Introduction 36

9.2 MRI_Format 36

9.3 Efficiency 36

9.4 Behaviour 37

10 Modeller 37

11 Renderer 38

12 MediaEngine 38

13 Scene 39

14 Interaction 42

14.1 Introduction 42

14.2 Input Device 42

14.3 Router 43

15 Coordinator 43

16 Functional Specification 45

16.1 Introduction 45

16.2 Non-object data types 45

16.3 Exceptions 46

16.4 Objects for coordinate spaces 47

16.4.1 Coordinate object 47

16.4.2 Colour object 48

16.4.3 TimeLocation object 48

16.5 Name object 49

16.6 Objects for media primitives 49

16.6.1 Primitive object 49

16.6.2 Captured object 50

16.6.3 Objects describing primitives with spatial and/or temporal form 50

16.6.3.1 Form object 50

16.6.3.2 Objects describing form primitives for audio media data 50

16.6.3.3 Objects describing form primitives for geometric media data 51

16.6.4 Objects describing primitives for the modification of media data 52

16.6.4.1 Modifier object 52

16.6.4.2 Objects describing modifier primitives for audio media data 52

16.6.4.3 Objects describing modifier primitives for structural aspects of media data 53

16.6.4.4 TimeFrame object 54

16.6.4.5 Objects describing modifier primitives for visual aspects of media data 54

16.6.5 Reference object 55

16.6.6 Objects for organising primitives into structures 56

16.6.6.1 Structured object 56

16.6.6.2 Aggregate object 56

16.6.6.3 Objects for organising media data within time 57

16.6.7 Tracer object 58

16.6.8 Wrapper object 59

16.7 Objects for describing properties of devices 59

16.7.1 MRI_Format object 59

16.7.2 EfficiencyMeasure object 60

16.8 Processing devices for media data 60

16.8.1 MRI_Device object 60

16.8.2 *Modeller* object 60

16.8.3 *Renderer* object 61

16.8.4 *MediaEngine* object 61

16.9 *Scene* object. 63

16.10 Objects for supporting interaction. 65

16.10.1 *InputDevice* object 65

16.10.2 *Router* object. 66

16.11 *Coordinator* object 67

17 Component Specification 69

A Overview of PREMO Modelling, Rendering and Interaction Object Types . 70

B Diagrammatic Conventions 73

B.1 Introduction 73

B.2 General Graphical Signatures 73

B.3 Conventions for Devices and Communication 74

C Relationship between Part 4 and the CGRM 75

C.1 Introduction 75

C.2 Architectural Links 75

C.3 Processing Links 76

C.4 Input and Output Primitives 76

C.5 Storage. 76

D A typical example scenario of MRI usage 77

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, government and non-governmental, in liaison with ISO and IEC, also take part in the work.

In the field of information technology, ISO and IEC have established a joint technical committee: ISO/IEC JTC1. Draft International Standards adopted by the joint technical committees are circulated to the national bodies for voting. Publication as an International Standard requires approval by at least 75% of the national bodies casting a vote.

ISO/IEC 14478-4 was prepared by Joint Technical Committee ISO/IEC JTC1, *Information technology*, Subcommittee SC24, *Computer graphics and image processing*.

ISO/IEC 14478 consists of the following parts under the general title *Information technology — Computer graphics and image processing — Presentation Environment for Multimedia Objects (PREMO)*:

- *Part 1: Fundamentals of PREMO*
- *Part 2: Foundation component*
- *Part 3: Multimedia systems services*
- *Part 4: Modelling, rendering, and interaction component*

Annex A forms an integral part of this part of ISO/IEC 14478. Annexes B to D are for information only.

Introduction

The Modelling, Rendering and Interaction component of PREMO describes facilities for the modelling and presentation of, and interaction with, multidimensional data that utilises multiple media in an integrated way. That is, the data may be composed of entities that can be rendered using graphics, sound, video or other media, and which may be interrelated through both spatial coordinates and time.

The objective of this component is to provide developers and users of modelling and rendering applications with a framework for supporting the definition and use of interoperable devices within a distributed setting. It achieves this by:

- a) providing an extensible framework of primitives for use in modelling, rendering and interaction which encompass multiple media, and which can be organized into larger structures and embedded into scenes.
- b) extending the resource and device hierarchies of the PREMO Part 3 (Multimedia Systems Services) Component to allow modelling, rendering and interaction to be uniformly integrated into a network of objects for managing the production and utilization of multimedia data.
- c) utilizing the property and capability management services of PREMO Part 3 to characterize the behaviour of modelling, rendering and interaction devices, allowing an application to be configured from such devices such that constraints on performance and functionality are satisfied.
- d) building on the object model and foundation objects of PREMO Part 1 and Part 2 to allow subsequent components to realize and extend specific modelling, rendering and interaction functionality.

This component follows PREMO Part3 in describing the external interface of object types and other entities involved in modelling, presentation and interaction, but not the internal structures needed to implement these. That is, it is not the purpose of this component to provide a set of building blocks that can be assembled into a modeller or a renderer. Rather, the component provides facilities to enable devices, built with various applications or performance trade-offs in mind, to interoperate in a heterogeneous presentation environment.

Information technology — Computer graphics and image processing — Presentation Environment for Multimedia Objects (PREMO) —

Part 4: Modelling, Rendering, and Interaction Component

1 Scope

This part of ISO/IEC 14478 describes a set of object types and non-object types to provide the construction of, presentation of, and the interaction with Multimedia information. The multimedia information can be graphics, video, audio, or other types of presentable media. This information can be enhanced by time aspects. Throughout this document this part of ISO/IEC 14478 will also be referred to as “Modelling, Rendering and Interaction”, and abbreviated as MRI.

The Modelling, Rendering and Interaction Component constitutes a framework of ‘Middleware’, system software components lying between the generic operating system and computing environment, and a specific application operating as a client of the services and type definitions provided by this component. It provides a framework over the foundation objects and multimedia systems services defined in other Parts of the international standard for the development of a distributed and heterogeneous network of devices for creating multimedia models, rendering these models, and interacting with this process.

The Modelling, Rendering and Interaction Component encompasses the following characteristics:

- a) provision of a hierarchy of multimedia primitives as an abstract framework for describing the capabilities of modelling and rendering devices, and for enabling their interoperability;
- b) within the primitive hierarchy, specific provision for describing the temporal structure of multimedia data through the stepwise construction of structured primitives from component data;
- c) provision of abstract types for modellers, renderers and other supporting devices, enabling the integration of such devices or any future subtypes representing real software or hardware, into a processing network of such devices;
- d) provision of an object type to map synchronization requirements expressed within multimedia primitives into control of the stream and synchronization mechanisms provided by ISO/IEC 14478-2 and ISO/IEC 14478-3.

The Modelling, Rendering and Interaction Component relies on the object types and services defined in PREMO Part 2: Foundation Components (ISO/IEC 14478-2), and PREMO Part 3: Multimedia Systems Services (ISO/IEC 14478-3).

2 Normative references

The following standards contain provisions which, through reference in this text, constitute provisions of this part of ISO/IEC 14478. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this part of ISO/IEC 14478 are encouraged to investigate the possibility of applying the most recent editions of the standards indicated below. Members of IEC and ISO maintain registers of currently valid International Standards.

ISO/IEC 11072:1992, *Information technology — Computer graphics — Computer Graphics Reference Model (CGRM)*.

ISO/IEC 7942-1:1994, *Information technology — Computer graphics and image processing — Graphical Kernel System (GKS) — Part 1: Functional description*.

ISO/IEC 9592:1997, *Information technology — Computer graphics and image processing — Programmer’s Hierarchical Interactive Graphics System (PHIGS)*.

ISO/IEC 14478-1:1998, *Information technology — Computer graphics and image processing — Presentation Environment for Multimedia Objects (PREMO) — Part 1: Fundamentals of PREMO*

ISO/IEC 14478-2:1998, *Information technology — Computer graphics and image processing — Presentation Environment for Multimedia Objects (PREMO) — Part 2: Foundation component*

ISO/IEC 14478-3:1998, *Information technology — Computer graphics and image processing — Presentation Environment for Multimedia Objects (PREMO) — Part 3: Multimedia systems services*

3 Definitions

3.1 PREMO Part 1 definitions

This part of ISO/IEC 14478 makes use of all terms defined in ISO/IEC 14478-1 (Fundamentals of PREMO).

3.2 PREMO Part 2 definitions

This part of ISO/IEC 14478 makes use of all terms defined in ISO/IEC 14478-2 (Foundation component).

3.3 PREMO Part 3 definitions

This part of ISO/IEC 14478 makes use of all terms defined in ISO/IEC 14478-3 (Multimedia systems services component).

3.4 Additional Definitions

For the purposes of this part of ISO/IEC 14478, the following definitions apply.

3.2.1 modeller: a virtual device that produces primitives on at least one output port.

3.2.2 renderer: a device that accepts primitives on at least one of its input ports.

3.2.3 media engine: a virtual device that accepts primitives from at least one of its input ports, and produces primitives on at least one of its output ports.

3.2.4 presentation: a collection of primitives that can be perceived by the operator.

3.2.5 coordinate: a primitive used to define a location in an nD space.

3.2.6 primitive: a structure describing information to be rendered, or information received through interaction.

3.2.6.1 form primitive: a primitive whose presentation has to be constructed by a renderer from an explicit description in terms of aspects or properties that characterize a class of perceivable representations.

3.2.6.1.1 geometric primitive: a form primitive used to define a shape or extent within nD space.

3.2.6.2 captured primitive: a primitive for which some or all of the perceivable aspects of the primitive have been encoded in some format defined externally to the PREMO standard.

3.2.6.3 structured primitive: a primitive that contains a collection of other primitives.

3.2.6.4 modifier primitive: a primitive that describes a change to the presentation of another primitive.

3.2.6.4.1 acoustic modifier: a modifier that changes properties of the sound generated by other primitives.

3.2.6.4.2 structural modifier: a modifier that affects the spatial and/or temporal aspects of another primitive.

3.2.6.4.3 visual modifier: a modifier that affects the (non structural) visual appearance of a primitive.

3.2.6.5 wrapper primitive: a primitive that carries a value drawn from some PREMO non-object data type.

3.2.6.6 tracer primitive: a primitive that carries an event for use in monitoring and coordinating the transfer of media data across a network.

3.2.7 input device: a device used to obtain data from the operator.

3.2.8 graphics: the construction, manipulation, analysis and presentation of pictorial representations.

3.2.9 scene: a device for storing and controlling access to a collection of primitive structures.

3.2.10 primitive structure: a collection of primitives organized into a structure that represents some or all of the data that describes a multimedia presentation.

3.2.11 coordinator: a MRI device that can manipulate the streams connecting its components.

3.2.12 router: a device for controlling the flow of data between streams connected to its ports.

3.2.13 temporal extent: the duration in time allocated or used for the presentation of some primitive.

3.2.14 operator: the external object that sends or receives information through a virtual device interface.

3.2.15 application: the external object or client that uses a PREMO system by creating objects, invoking operations on objects, and using types defined by PREMO. Applications are not modelled in the PREMO system, but their interactions with a PREMO system are modelled.

The following alphabetical list gives the sub-clause of each definition.

acoustic modifier	3.2.6.4.1
application	3.2.15
captured primitive	3.2.6.2
coordinate	3.2.5
coordinator	3.2.11
form primitive	3.2.6.1
geometric primitive	3.2.6.1.1
graphics	3.2.8
input device	3.2.7
media engine	3.2.3
modeller	3.2.1
modifier primitive	3.2.6.4
operator	3.2.14
presentation	3.2.4
primitive	3.2.6
primitive structure	3.2.10
renderer	3.2.2
router	3.2.12
scene	3.2.9
structural modifier	3.2.6.4.2
structured primitive	3.2.6.3
temporal extent	3.2.13
tracer primitive	3.2.6.6
visual modifier	3.2.6.4.3
wrapper primitive	3.2.6.5

4 Symbols and abbreviations

CGRM:	Computer Graphics Reference Model.
CSG:	Constructive Solid Geometry
GKS:	Graphical Kernel System.
IEC:	International Electrotechnical Commission.
ISO:	International Organization for Standardization.
MPEG:	Moving Picture Experts Group.
MRI:	Modelling, Rendering and Interaction
MSS:	Multimedia Systems Services
PHIGS:	Programmers Hierarchical Interactive Graphics System.
PREMO:	Presentation Environment for Multimedia Objects.
VRML:	Virtual Reality Modeling Language.
nD:	Multi-dimensional.
2D:	Two-dimensional.
3D:	Three-dimensional.

5 Conformance

A conforming implementation of the PREMO Modelling, Rendering and Interaction Component shall comply with the general conformance rules defined in clause 5 of ISO/IEC 14478-1 and the component specification in clause 16.

6 Overview of the Modelling, Rendering and Interaction Component.

6.1 Introduction

This clause presents an overview of the modelling, rendering, and interaction component (Part 4) of PREMO. It summarises the concepts defined in the document, and explains how these concepts contribute to the goals set out in the Introduction. More detailed descriptions of the concepts used in the overview are given in subsequent clauses. This part of ISO/IEC 14478 also makes extensive use of facilities provided by PREMO parts 1-3, in particular the device and stream concepts introduced in ISO/IEC 14478-3. A summary of these dependencies is included in this clause.

6.2 Overview

The model underlying this part of ISO/IEC 14478 is that a multimedia system consists of modellers, renderers, and other devices (some media specific) linked together via streams that carry data of a particular format. These concepts of stream and device are those defined in ISO/IEC 14478-3. A device consists of a processing facility, together with a number of ports through which it can accept input and produce output, using a format defined by the port. Figure 1 shows a high-level view of a (simplified) example system in which a graphical user interface is used to control parameters of an audio-visual presentation system. Rectangles on the sides of devices represent ports, and the thick lines between such ports represent media flow via streams. Thin lines represent other forms of interaction, for example operation invocation. See also Annex B for a table defining the symbols used in the figure. The system in Figure 1 consists of:

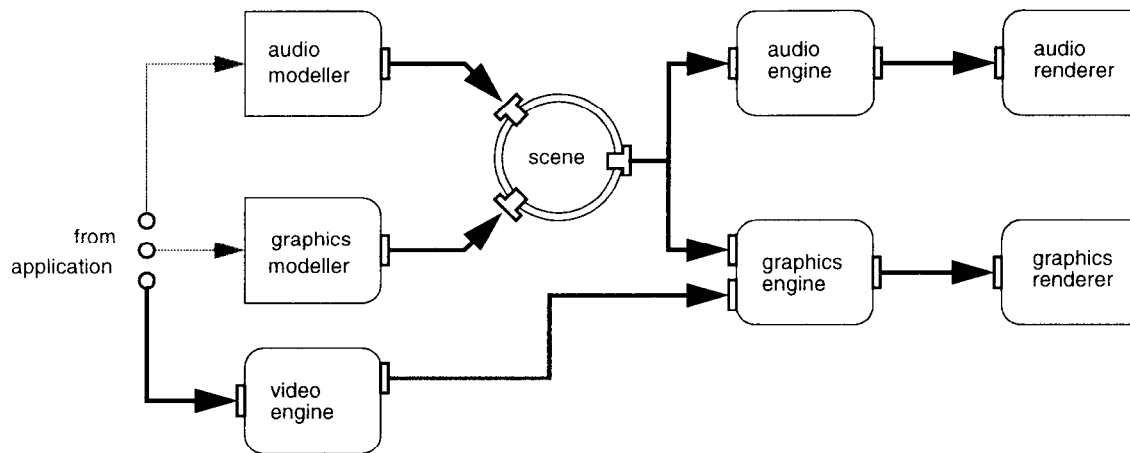


Figure 1 — An audio/visual system

- two modellers, one for audio data and one for graphical data, that might be used to construct and edit primitives via an interface to the application specific to the needs of each modeller;
- a video engine, accepting a stream of video data from the application and constructing a stream of information in some video format;
- a scene, which encapsulates primitives produced by the two modellers and which mediates access to the collection of primitives by both the modellers and the associated renderers;
- an audio engine, that takes primitives from the scene and converts into a data format that can be used to drive an audio-specific device;
- a graphics engine (acting as a mixer or composition tool) uses the video output from the video renderer and primitives from the scene to construct a further stream of primitives, integrating the two sets of source data within some appropriate visual representation. The primitives produced by the engine may be some subset suitable for input to a specific renderer;
- two renderers, one for audio and one for graphics, that convert a stream of primitives into a form that can be processed in the context beyond the MRI network (in this case, presented to the end user of the application).

The figure shows just one way in which such a system might be implemented. A different implementation may collapse the three engines into a single device, if it has access to a media engine that can take both graphical and audio primitives as input and that can generate the corresponding output streams. Another implementation may decompose the graphics engine into, for example, a number of components that manage specific functionalities such as viewing or clipping. Finally, a high-performance implementation might collapse all components into a single device.

Key components of this part are derived from the object types defined in ISO/IEC 14478-3, in particular the property inquiry and constraint facilities. These types and facilities may be used by the factory mechanism described in ISO/IEC 14478-2 to produce objects that meet certain requirements, and by the negotiation and QoS mechanisms for establishing and maintaining a network of objects that satisfies specific properties. As a result, many of the object types defined in this part have an associated list of properties for use in creation and negotiation. For example, modellers and renderers are derived (indirectly) from the *VirtualDevice* type of ISO/IEC 14478-3, and thus inherit the Property Inquiry services. Each renderer and modeller has a collection of properties that characterise its capabilities in terms of its inputs, outputs and quality of service. An application using the facilities of this part can request creation of a renderer from a renderer factory by invoking the *'create_object'* operation using the type 'renderer' as the name of the object type and passing a structure containing the required capabilities. Alternatively, an application that is aware that the rendering interface it requires is defined by a specific subtype of the renderer can request the factory to produce a renderer of that specific type.

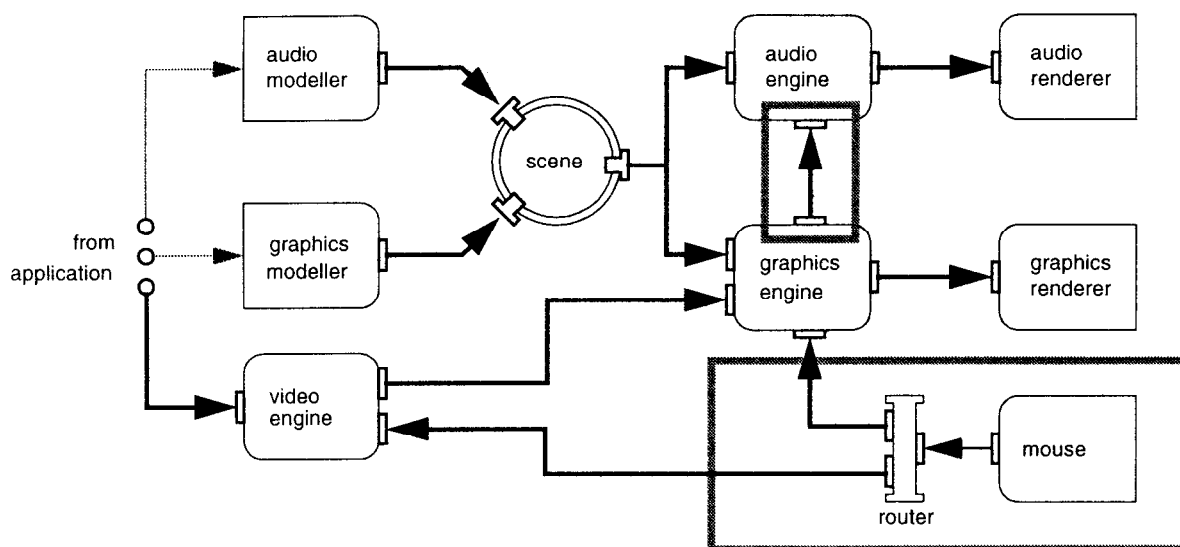


Figure 2 — MRI network including interaction handling

For simplicity, Figure 1 illustrates devices used in presentation only. The same approach of using specialised virtual devices connected by data streams is employed for handling input and interaction. Figure 2 extends the example with a simple framework for input handling. This introduces two new devices, and a new information flow between two renderers, shown in the outlined regions of the figure.

- The mouse is an example of an *InputDevice* that can provide primitives for processing elsewhere in the network, either via a stream (as shown) or through a procedural interface or callback mechanism.
- A *Router* can be incorporated to allow a data stream to be sent to specific devices depending on some internal state. The *Router* device achieves this by also subtyping from the *Controller* object defined in ISO/IEC 14478-2.
- Although *Engine* objects primarily operate on streams of primitives intended for presentation, as the example shows, an engine may also have ports that are used to receive (and in the case of the graphics engine, to transmit) primitives used to carry data about input.

Figure 1 and Figure 2 focus on the main streams and virtual devices involved in a simple MRI network. In addition to these, ISO/IEC 14478-3 provides object types for establishing and controlling a collection of streams and devices. These types are called *VirtualConnection* and *Group*, respectively.

An instance of *VirtualConnection* is an object that represents an abstract view of media transport between devices, allows control over aspects of the connection, and is responsible for negotiating the connection in terms of formats and quality of service considerations etc. Several kinds of virtual connections are possible, depending on whether the devices have compatible ports, and on whether a particular connection is local or networked. General examples of these are given in Annex D of ISO/IEC 14478-3. If the ports of two devices are not compatible, or the devices are in different parts of a distributed environment, *connection adaptors* will be employed. These adapters are an implementation concept, not visible to the application, and are not defined as an object type in the profile of ISO/IEC 14478-3.

Independent from issues of distribution and the existence of virtual connections, the *Group* object type provides applications with the ability to manage a collection of objects that are instances of *VirtualResource* or its subtypes (e.g., devices and connections) as a single resource. As *Group* is subtyped from *VirtualResource*, this arrangement can be hierarchical. Groups provides facilities to acquire the resources needed to establish a number of connections, to monitor the end-to-end quality of service, and to provide an application with a single access point for monitoring and controlling the flow of data across the resources that make up the group.

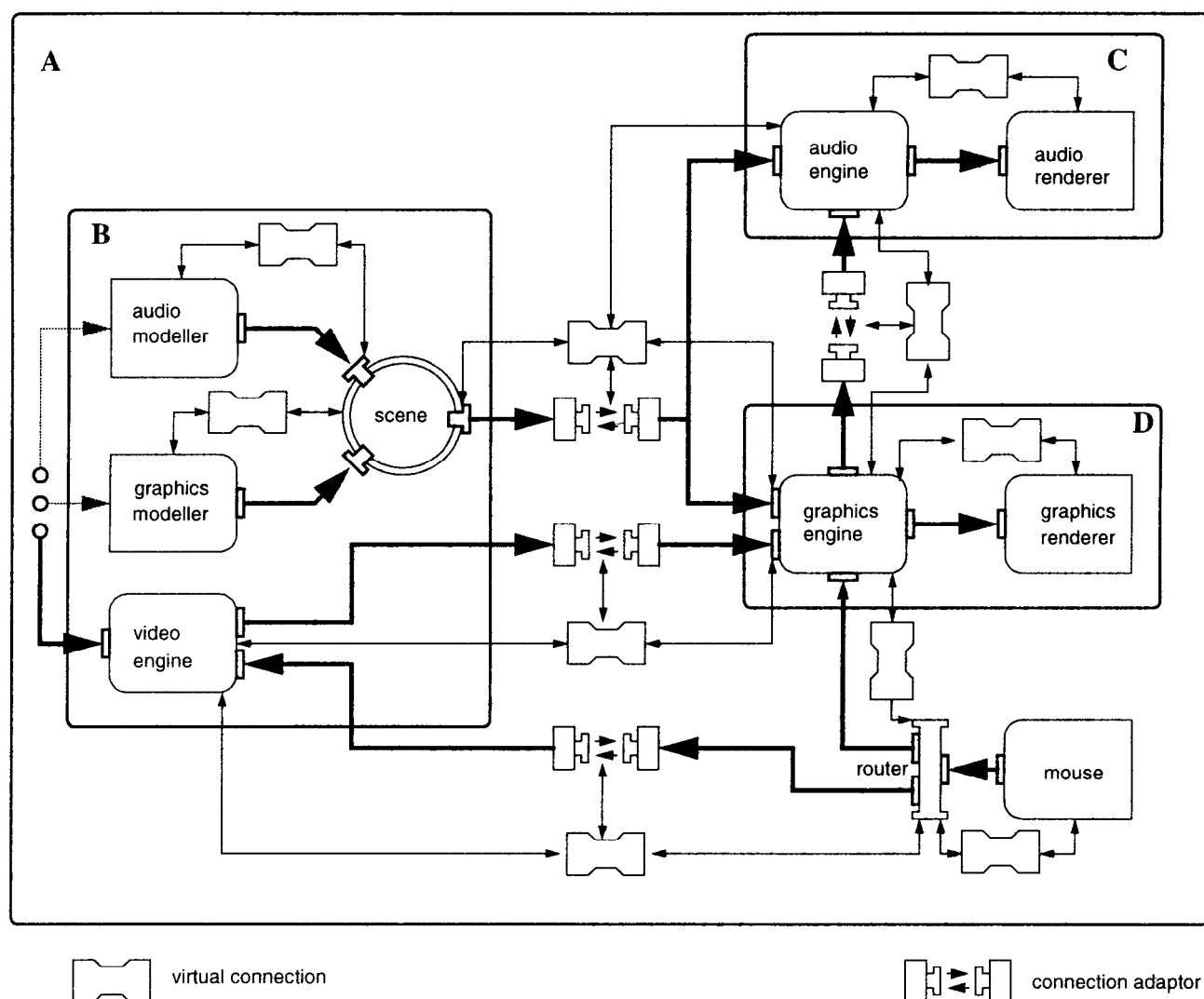


Figure 3 — Groups and connections in audio/visual system

Figure 3 shows one arrangement of groups could be used to implement the audio-visual rendering example. It also illustrates, for completeness, where connection adaptors may be required. In considering distribution, it has been assumed that the following sets of components are each located at separate nodes of a distributed system:

- a) the modellers, the video engine, and the scene;
- b) the audio engine and renderer;
- c) the graphics engine, renderer, mouse, and router.

With respect to groups, there are a number of possible arrangements for the network. One such arrangement, consisting of four groups (labelled A-D) has been shown in the figure. In more detail,

- group A is the outer(most), and contains the other three groups, the mouse and router, and the connections and adaptors used to link devices in Groups B, C and D;
- group B contains the modellers, the video engine, and the scene, plus the virtual connections needed to link devices within the group;

- group C contains the audio engine and renderer, and the virtual connection between them;
- group D contains the graphics engine and renderer, and the virtual connection between them.

Although these groups in the example are loosely based around the devices located at each node there is no assumption or requirement in PREMO that there should be such a correspondence. Groups are a logical concept, and in general are independent of the system organisation. Although users of this part of ISO/IEC 14478 may need to be aware of groups and connections and may wish to make use of them explicitly when constructing a MRI network, ISO/IEC 14478-3 also provides the *LogicalDevice* object types that can be used to hide much of this detail from an application. Furthermore, for applications that distribute presentation data (e.g. primitives, or other media-specific formats) to multiple renderers or devices, the use of *Coordinator* devices may also be necessary. The *Coordinator* object type is defined in this part of PREMO, and is a subtype of *MRI_Device*. Its role is described in 6.7.

The purpose of this overview is to illustrate some of the key connections between the object types defined in this part and those introduced by ISO/IEC 14478-1 to ISO/IEC 14478-3. In practice, the object types mentioned in the discussion will only be a subset of those utilized by an application. For example, in the audio-visual scenario, mouse input is passed along a stream to the graphics renderer, which may be providing immediate visual feedback on the changes made to certain control parameters (perhaps by adjusting the drawing of interface elements showing the volume, speed etc. of the audio and video playback). Other mechanisms, in particular event handlers and reference points, may also be present to control or mediate interaction between the components of the system, but have not been mentioned in this overview.

6.3 Devices for Modelling, Rendering, and Interaction

This part of PREMO introduces a hierarchy of devices for processing streams of media data constructed using objects based on the primitive hierarchy described elsewhere in this part (see 6.4). The root of this local hierarchy is the *MRI_Device* object type that extends the *VirtualDevice* object type defined in Part 3. This means that the devices defined in this part as subtypes of *MRI_Device* can be directly integrated into a network of more general devices that may include media-specific input and output devices as well as more abstract processing nodes. An example of such a network has been developed in 6.2. As virtual devices, *MRI_Device* and its subtypes contain a number of ports that allow either input or output of data in a particular format. The *Format* object type is introduced in ISO/IEC 14478-3, and a small subtype hierarchy for certain common media formats is presented as an informative annex (Annex C) of that part. The Modelling, Rendering, and Interaction component defines a subtype of the *Format* object type, called *MRI_Format*, that characterises a data stream that carries the primitives described in 6.4. Subsequent PREMO components, or applications, may specialize this format object type to define the input and output format of a MRI device that can utilise a richer collection of primitives. Figure 4 illustrates the relationship between the *Format* and *MRI_Format* object types, the examples of *Format* object types from Annex C of ISO/IEC 14478-3, and any future extensions for processing specialised streams of MRI primitives.

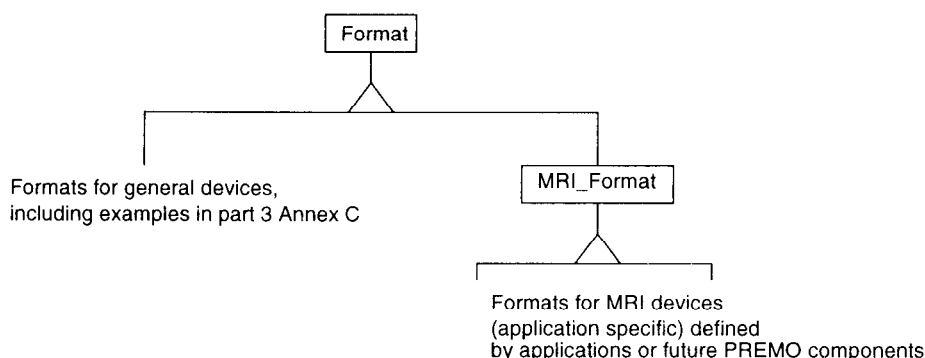


Figure 4 — Format hierarchy

This part defines a number of subtypes of *MRI_Device* as abstract object types for particular kinds of processing functionality. Three of these generalize the traditional computer graphics concerns, of modelling and rendering, to multimedia data processors. These devices differ in the kind of interface that they are required to support for input and output.

- A *Modeller* is a device that provides at least one output port that can support *MRI_Format*. In contrast, the input interface of a modeller is typically application-specific. A CSG modeller might offer operations that allow the construction and

manipulation of a CSG solid, while a modeller for music scores defined using audio primitives might have a direct manipulation interface through which an end user interactively constructs the score, which the modeller can then pass as a stream of primitives to other MRI devices for processing.

— A *Renderer* is a device that provides at least one input port that can use *MRI_Format* to accept primitives. The output interface of a renderer is typically application specific. Renderers include devices that output primitives directly to end users as a presentation (images, sound etc.) as well as devices that drive further software or hardware components of a system through some interface, the nature of which is not of concern to this part. This does not preclude a specific renderer from utilising PREMIO functionality for its output mechanisms.

— A *MediaEngine* is both a *Modeller* and a *Renderer*. A *MediaEngine* is a device that offer at least one input port and at least one output port that are capable of operating with *MRI_Format*, allowing the engine to accept, process and produce streams of primitives. Media engines, like other MRI devices, may have additional ports that allow the device to accept or produce data in other formats.

NOTE — If a media format such as MPEG is to be taken directly as input by a renderer, as for example in the audio/visual system shown in Figure 1, then the raw input stream may need to be converted into a stream of primitives in which the MPEG data is carried within the primitive structure described shortly. This will allow additional information to be attached, such as a geometric data to characterise the position of the video data within a graphical scene. This transformation may either be a capability of a renderer, or may be realized by a specific device that acts as a translator or pre-processor. Such translators could be introduced by subsequent components, if any, as an object type derived from *MediaEngine*. A translator would have exactly two ports, one accepting a stream of an arbitrary format, the other generating a stream of primitives, with the semantics that the translator simply ‘wraps’ the media stream into an appropriate primitive object.

6.4 Primitives and Coordinates

6.4.1 Introduction

The MSS component, which provides the base object types from which MRI devices are derived, contains no statement on the content and structure of the data carried on streams and operated on by virtual devices. The contribution of the MRI component is to add just such information about the data used to realize multimedia presentation and interaction. This data, which is described by a hierarchy of PREMIO object types, are called *primitives*. The organization of primitive types into categories or hierarchies varies widely in practice. This part of ISO/IEC 14478 defines primitive object types where it is intended that client applications, or any subsequent components that address the needs of specific domain areas or presentation technologies, extend this hierarchy in a way most appropriate to their specific needs. By providing a common set of basic primitive types, this part of PREMIO is able to utilize the property and negotiation frameworks defined in ISO/IEC 14478-2 and ISO/IEC 14478-3 for establishing a network of MRI devices, and ensures that there is common vocabulary for basic MRI device capabilities.

6.4.2 Coordinates

Primitives within the PREMIO hierarchy that address specific media such as graphics and audio are defined as abstract object types to ensure that they can adequately be specialized to the needs of specific application areas or implementations. As such, primitive object types do not define the structure of primitives in terms of coordinates within the various spaces (cartesian or other geometric spaces, properties of sound, etc.) through which primitives are specified in practice. However, to support inter-operability between implementations of the primitive hierarchy, this part of ISO/IEC 14478 defines a generic object type for coordinates within spaces of arbitrary dimensionality, subject to the constraint that the components representing each dimension of a coordinate are each drawn from the one PREMIO type.

A specialization of this notion of coordinate, to the representation of colours within a colour model, is also provided by this part. A mapping from four colour models into the dimensions of the colour object type is provided.

6.4.3 The Primitive Hierarchy in PREMIO

6.4.3.1 Overview

Primitives are structures that primarily carry information that is to be rendered to the user, or information that characterises input. PREMIO does not attempt to describe a closed set of primitives for modelling and rendering. Instead, the approach of this part is to provide a general, extensible framework that provides a uniform basis for deriving primitive sets appropriate to specific appli-

cations or renderer technologies. In general, modellers may use specific techniques such as Constructive Solid Geometry or NURBS surfaces for a particular range of applications. Such techniques may require an enriched set of basic primitives. The aim of the primitive hierarchy defined in this part is to provide a minimal common vocabulary of structures that can be extended as needed, either by applications using PREMO, or by subsequent PREMO Components providing modellers and renderers to utilise a specialized set of primitives. For example, a component for Constructive Solid Geometry may enrich the primitive hierarchy with object types to represent a CSG tree, for example in the form of object types for basic geometric solids together with Intersection and Union types.

More specifically, the output generated by modellers and renderers, and the input to renderers, can consist of structures that are derived from a type called *primitive*. Examples of the information that might be carried by a primitive include:

- a polyline (in the sense of GKS and PHIGS);
- a Bézier curve;
- a triangular mesh;
- MPEG data;
- an audio waveform envelope located at a spatial coordinate;
- a location on the display (obtained by a PICK device).

At the top level, the primitive hierarchy consists of seven categories.

6.4.3.2 Captured Primitives

A captured primitive contains a reference to a source of raw data encoded in some standard format such as JPEG, MPEG, MIDI, or VRML. This data may happen to be recorded (for **example**, an image stored as a JPEG file, or a video stored as an MPEG file), or live, for example fed from a microphone device.

6.4.3.3 Form Primitives

This category represents primitives whose presentation has to be generated in some way, for example lines, curves, sounds, or textual information. This type is further classified into types representing specific kinds of information. The text below lists the different kinds of form primitive and provides some examples of their use - note that the examples do not necessarily correspond to subtypes defined in this part of ISO/IEC 14478. Rather, they illustrate how the hierarchy might be utilized in any subsequent components or applications of PREMO. Full details of the primitive types in PREMO are given in clause 8.

- Audio: information for generating sounds;
- Geometric: lines, planes, surfaces, curves, point data sets etc.;
- Tactile: a description of haptic properties;
- Text: character strings to be represented in some way.

6.4.3.4 Modifier Primitives

A modifier primitive carries information that affects the presentation of other primitives. Examples include visual effects (colour and texture), transformations on the coordinates that define the structure of media data, and audio effects.

- Acoustic: sound effects, and characteristics of voices;
- Structural: transformations on coordinates, such as scaling and rotation in geometric space, operations on time; constraints for use, for example, in geometric and/or temporal clipping are also subtypes of Structural modifier;
- TimeFrame: changing the clock against which progression of time is measured;
- Visual: properties of light, texture information etc.

6.4.3.5 Reference Primitives

A reference primitive allows some part of a primitive structure to be reused, by providing a means of referring to that part of the structure in terms of strings carried by a *Name* object. *Name* objects are introduced into a presentation by structured primitives.

6.4.3.6 Structured Primitives

A 'structured primitive' is one that contains a collection of other primitives (which may including further structures), and a name by which that structure can be referenced. By naming structured primitives, it becomes possible to refer to part of a primitive structure from elsewhere in that structure, thereby facilitating reuse. Such names can also be used in application-dependent ways to provide feedback, for example on structures selected during interaction. PREMO identifies two kinds of structured primitive, each with a specific semantics:

- *Aggregate*: An aggregate is a combination of primitives in which information carried by some primitives is intended to modify the presentation of others in the aggregate. For example, a coloured line might be defined by aggregating a polyline (geometric) primitive with a visual colour modifier. Similarly, a sound effect might be applied to a recording referenced through a 'captured' primitive, while a polygonal mesh might be defined via some geometric primitive aggregated with specific colour, linestyle and linewidth modifiers.
- *TimeComposite*: Multimedia presentations exist in time, and may involve the coordinated presentation of different streams of data. A *TimeComposite* primitive defines the way in which the presentation of its components are to be related in time. PREMO defines three kinds of primitive that organise the components of *TimeComposite* to achieve specific effects.
 - Sequential; one primitive follows another.
 - Parallel; the temporal extents of the primitives overlap.
 - Alternative; there is a choice between components, depending on the current state of a controller object.

NOTE — Any of the primitives that appears as a component of a structured primitive may itself be structured, i.e., may be a *TimeComposite* or *Aggregate* primitive. Consequently it is possible to construct complex multimedia presentations from simpler elements by organising them into a hierarchy that approximates the required pattern of coordination. Fine control over synchronization can then be effected through other mechanisms, such as the use of synchronization points.

6.4.3.7 Tracer Primitives

A tracer primitive carries an event object. When such a primitive is encountered at the input or output port of a device derived from the *MRI_Device* object type, the event handler attached to that port will be notified of the event carried by the primitive. The use of tracers in supporting coordination between devices is explained in clause 13.

6.4.3.8 Wrapper Primitives

This class of primitive allows arbitrary PREMO values to be carried as primitives, for example for use in handling interaction.

6.4.4 Primitives and MRI Devices

The semantics of the primitive hierarchy with respect to devices for media processing are defined in terms of the diagram in Figure 5. The two parts of the figure show part of a primitive hierarchy, and a renderer accepting a stream of primitives via a port with a format that is defined to accept certain kinds of primitive. A sample input and output stream is also shown, with the 'effect' of rendering indicated (for illustration) by the application of a pattern to the primitives. The renderer in Figure 5(a) accepts two types of geometric primitive, P1 and P2, that are defined as subtypes of *Geometric* in the corresponding type hierarchy. If the type hierarchy is subsequently extended as shown in Figure 5(b), by deriving new geometric primitives P3, P2.1 and P2.2, and the format attached to the renderer's input port is modified to allow the port to accept primitives of type P3, then the following behaviour will result.

- a) primitives of type P1 and P2 will be rendered as in part (a) of Figure 5.

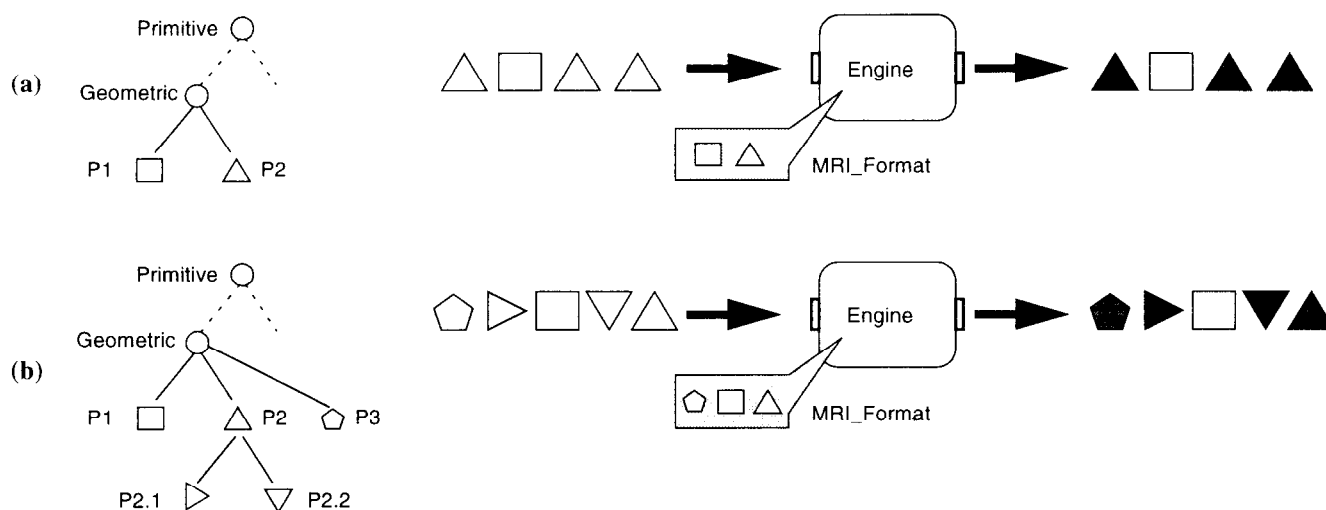


Figure 5 — Media devices and the primitive hierarchy

b) primitives of type P3 (pentagon) will be rendered in a way defined by the renderer, as these have been included explicitly in the list of primitive types that the renderer can accept via that port.

c) primitives of type P2.1 and P2.2 (drawn as rotated triangles) will be rendered as if they were primitives of type P2. These primitives are subtypes of a primitive type (P2, shown as a triangle) that the renderer can accept through the input port, but as the specific details of these subtypes are not explicitly known to the renderer, it will treat instances of P2.1 and P2.2 as instances of P2.

If a primitive p is presented to a *Renderer* or *MediaEngine* via a port, and the type of p is not a subtype of any of the primitive types that are defined for that port, then the device will raise an exception, *PrimitiveNotRecognised*. Apart from this exception, the behaviour of a MRI device on receipt of an unrecognized primitive is application dependent.

6.5 Scene

A simple PREMO application may realise a rendering network by connecting directly output ports of one device (for example a modeller) to the input ports of other devices, such as renderers. However, this simple strategy is not always appropriate, particularly in the kind of high-end application areas for which PREMO is intended. For example, there may be more than one modeller contributing primitives for presentation, or there may be multiple renderers at different locations that want to render a subset of the primitive structure. To support these needs, this part defines an object type called *Scene*, instances of which can be placed into a modeller/renderer network to act as a storage container and access mediator for a shared primitive structure. As a scene is also a kind of *VirtualDevice* it can be inter-connected freely with other devices in the modelling and rendering network.

The *Scene* object type provides operations that allow applications to construct, access, and delete a number of *primitive structures*. These structures are built by organising smaller primitives (or primitive structures) into larger units through the various structuring mechanisms provided by this part of ISO/IEC 14478 or by new mechanisms introduced by subsequent components. A primitive structure represents information that will form some or all of a multimedia presentation. Access to primitive structures within a scene is effected through the use of a *name*-valued attribute associated with the *Structured* primitive object type and its subtypes.

6.6 Interaction

Interaction refers to the ability of system components to accept and respond to input. In terms of the CGRM (see Annex C), this is information that originates as input from the operator to the Realization Environment and is then sent towards the Construction Environment. The input can be processed at various points along the pipeline, for example to provide feedback, or can be passed to further components of the modelling/rendering network. Input data is realized as instances of object types in the *Primitive* hierarchy described in 6.4. That is, PREMO does not impose a separation between primitives used for output and those used for input. As this part does not impose any specific structure on devices for modelling, rendering and interaction, there is also no explicit provision of mechanisms for the control of interaction or feedback. Exactly how feedback can or should be handled will depend on the configuration of specific devices and the combination of modalities that are employed by a given application.

Interaction in PREMO is effected through the use of objects that are instances of *InputDevice*, an object type derived from *Modeller*. As an input device is also indirectly an *MRI_Device* and a *VirtualDevice*, it contains, in addition to a procedural interface, a reference to an event handler and ports for stream data. Through these various mechanisms, PREMO supports the construction of three fundamental approaches to input handling: event, sampled and request modes.

- a) *Event mode*. A client that is interested in receiving input from the device registers itself with the event handler attached to the processing element of the device. This will generate an event when new data are available, and will pass the data to registered objects via events.
- b) *Sampled mode*. A client is connected to the device by a stream which is initially muted. When the client wishes to sample input, the stream is put into play mode until the next item of data arrives; it is then returned to mute.
- c) *Request mode*. A client invokes the operation request in the interface of the input device. This operation is synchronous, and therefore control is transferred to the device until the operation returns. The type of the datum returned by the operation is a subtype of *Primitive*.

That is, in 'event mode' the event handler is used to return input data, in 'sampled mode' the client utilises the output stream, and in 'request mode' the input data is returned as a result parameter of the *request* operation.

Management of event mode interaction is handled by the use of *EventHandler* object types and subtypes defined in ISO/IEC 14478-2, and no further mechanism is defined in this part. For sampled mode interaction, this part introduces the concept of a *Router* object to assist applications in coordinating the distribution of input data. A router is a virtual device that is also subtyped from *Controller*, and thus combines a collection of input and output ports with a finite state machine. Each state of a router is characterized by the connections from input ports to output ports. This assignment of data flow to states is dynamic, and can be changed by the application or client using operations in the interface of the *Router* object type.

6.7 Coordinators

The primitive hierarchy of PREMO, in particular the *TimeComposite* object type and its immediate subtypes, forms the basis of a declarative model of multimedia presentation content. That is, a collection of primitives, organised into a hierarchy using various kinds of *Structured* primitive, represents a static description of a presentation. For this presentation to be realised, i.e. displayed to a user or otherwise processed, its component primitives must at some point be operated on by a suitable device, i.e. a device that can accept this kind of primitive on its input port. Also, if the presentation consists of parallel components (in the form of a *Parallel TimeComposite* primitive, for example), then either a device has to be found that can manage parallel presentation, or two or more devices have to be used in parallel within a processing network. In the latter case, some mechanism is required to enable synchronization between these devices. These two requirements — the allocation of primitives to suitable media devices, and the scheduling and synchronization of parallel media content — are significant technical problems in the design of a multimedia system. While ISO/IEC 14478-1 to ISO/IEC 14478-3 do not mandate any one solution, this part of ISO/IEC 14478 defines an object type, called *Coordinator*, that can provide the behaviour described above.

The *Coordinator* object type is a subtype of *MRI_Device*. Each instance of this type has exactly one input port on which it can receive a stream of primitives in *MRI_Format*. Its interface contains operations that allow a specified device to be added to or removed from the set of devices that the coordinator can use in processing its own input stream. These facilities are similar to those provided by the *Group* object type defined in ISO/IEC 14478-3, but are restricted in comparison with those of the *Group* object type. Also, when a device is added to a coordinator, it is necessary to specify the input port of the device to which the coordinator is to direct appropriate primitives. For these reasons, *Coordinator* is not a subtype of *Group*.

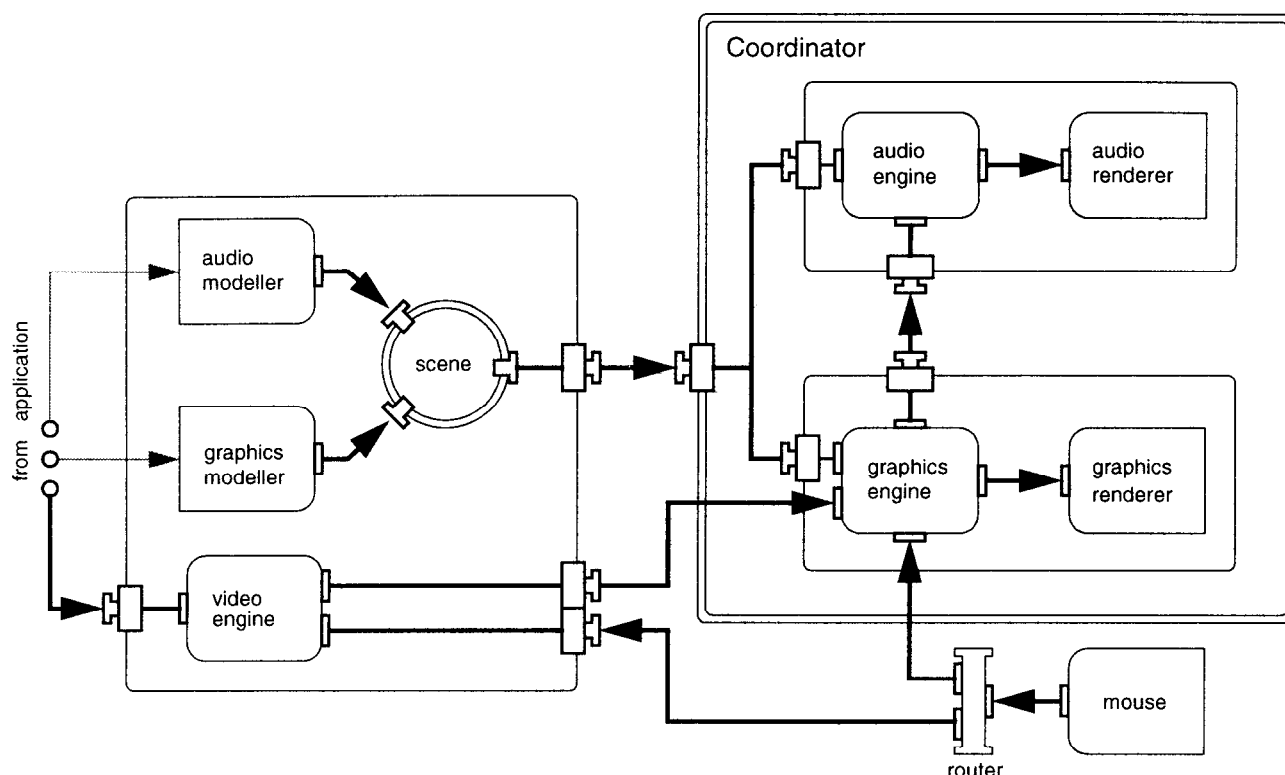


Figure 6 — Audio/visual system using logical devices and coordinator

The devices that can be added to or removed from a coordinator are required to be subtypes of *Renderer*. However, functionality that is not directly available within a single *Renderer* can still be utilised by a coordinator provided it is encapsulated in a type that inherits from *Renderer*. For example, an application that wishes to make a network of devices available to a coordinator could define a new type that inherits from *Renderer* and the *LogicalDevice* object type defined in ISO/IEC 14478-3. The operations inherited from *LogicalDevice* would allow the network to be established and viewed as a single device, while inheriting from *Renderer* ensures that objects of this new type can interoperate with a coordinator. Figure 6 shows how, by using such an approach, the system first illustrated in Figure 1 might be realised using three devices derived from *LogicalDevice* (labelled A, B, and C), and an instance of *Coordinator*. The coordinator encapsulate the two devices concerned with rendering, which will assume are also subtypes of *Renderer*. The presentations accessed from the scene object arrive at the coordinator, where they are decomposed into components suitable for the audio engine and renderer, and for the graphics engine and renderer. Note that the graphics engine can obtain data directly from another device, by-passing the input port of the coordinator.

As a coordinator has access to the devices to which it distributes media primitives, it has the potential to manage synchronization between these devices by manipulating the *StreamControl* object associated with the processing element of each device. However, other approaches are possible, either as alternatives to, or in parallel with, this scheme. For example, a coordinator could use an *ANDSynchronizationPoint* object, and reference points on the input streams to its component devices, to enforce synchronization constraints. The could also utilise *Tracer* primitives to monitor the progress of media data through the various processing elements. Given the range of possible synchronization mechanisms, ISO/IEC 14478 does not mandate that a coordinator implement any specific scheme. Instead, it is expected that particular synchronization strategies will be realized by subtypes of the *Coordinator* object type defined by specific applications or any future PREMO component that extends this part of ISO/IEC 14478. Further details on the role of the *Coordinator* object type in media synchronization can be found in clause 15.

6.8 Dependencies on other Parts

This part of ISO/IEC 14478 builds extensively on the services and object types defined in other Parts of the standard. At a definitional level, it utilises the object model and notational conventions described in part 1. The creation and operation of an MRI system is also expected to utilise the Property Management services and Factory mechanisms set out in part 2. However, there are also some specific dependencies, where object and non-object data types defined in ISO/IEC 14478-2 and ISO/IEC 14478-3 have been used directly in defining the types and services of this part. These specific dependencies are listed below.

- The *Controller* object type defined in ISO/IEC 14478-2 is used as a supertype for the *Router* object type. Each *Alternate* primitive also contains a reference to a *Controller* instance.
- A *TimeComposite* primitive contains a reference to an instance of the *EventHandler* object type defined in ISO/IEC 14478-2.
- A *TimeFrame* primitive contains a reference to an instance of the *Clock* object type defined in ISO/IEC 14478-2.
- Instances of the *Event* structure defined in ISO/IEC 14478-2 are created by *TimeComposite* objects during processing.
- The *Time* non-object data type defined in ISO/IEC 14478-2 is used in *Temporal* and *TimeComposite* primitives.
- The *Port* non-object data type defined in ISO/IEC 14478-3 is used in the definition of *Captured* primitives, and in the definitions of the *Scene* and *Router* object types.
- The ISO/IEC 14478-3 *Format* object type is the superclass for the *MRI_Format* type.
- The *VirtualDevice* object type defined in ISO/IEC 14478-3 is the supertype of the *MRI_Device* object type defined in this part.
- The *Primitive*, *Coordinate*, *EfficiencyMeasure* and *Name* object types defined in this part are immediate subtypes of the *SimplePREMOObject* object type defined in ISO/IEC 14478-2.

6.9 Subtyping Diagram

Figure 7 on page 24 gives an overview of the object type hierarchies introduced in this part, and in particular their relationship to the object type hierarchy defined in ISO/IEC 14478-1 to ISO/IEC 14478-3. Object types introduced in this part are shown with their names enclosed in boxes. The hierarchy of object types subtyped from *Primitive* has been elided from this diagram, but can be found in Figure 8 on page 26. Also, full object-type diagrams can be found in Annex A. Note that the *Router* and *MediaEngine* object types are defined through multiple inheritance.

7 Coordinates

7.1 General Coordinates

PREMO is concerned with audio and video, as well as with synthetic graphics, tactile or other media. These data may be located within specialised coordinate spaces (e.g. frequency, pressure, etc.) and thus it is necessary for this part of PREMO to include a general, extensible representation of coordinate. To this end, a generic *Coordinate* object type is defined with the following interface:

- a *dimensionality* (read only), i.e. the number of dimensions;
- a *getRange* operation which, for a given dimension, returns the range of values allowed in that dimension; and
- *setComponent* and *getComponent* operations which access and update the *i*th component of the coordinate, where $1 \leq i \leq \text{dimensionality}$.

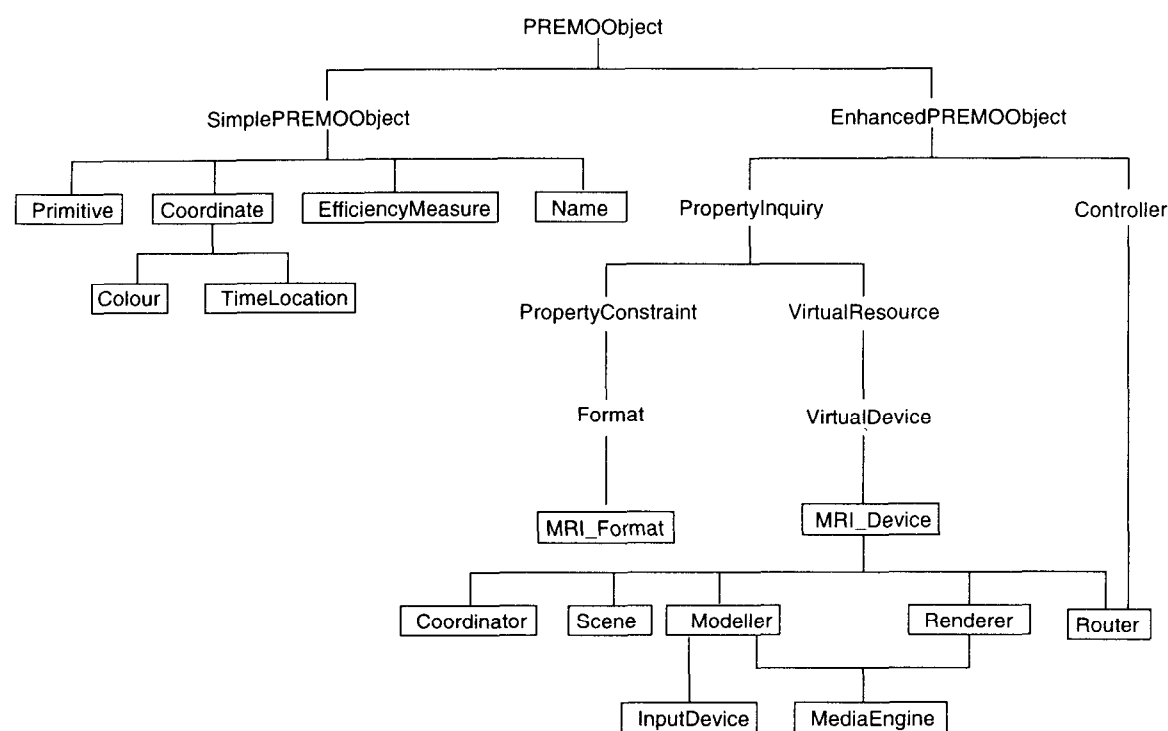


Figure 7 — Subtyping diagram

The coordinate object type is generic to allow the types from which components of coordinates are drawn to vary. However, for any given coordinate object, all components of that coordinate will be from the actual type used to instantiate the generic parameter. That is, each component of a coordinate must belong to the one type.

NOTE — To support efficient implementation of coordinates, the effect of accessing (via *setComponent* and *getComponent*) a component of a coordinate outside the dimensionality of that coordinate is not defined. Similarly, this part of ISO/IEC 14478 does not require that the value stored in the component of a coordinate be checked against the specified range for that component. Such checks can be implemented by subtyping the *Coordinate* object type and redefining the operations, for example to raise an exception when components or values outside of the specified range are supplied as parameters.

7.2 Colour

This part of PREMObject defines an object type, *Colour*, as a subtype of *Coordinate* specialized to represent colours within a range of colour models in use. In this object type, the dimensions of the generic coordinate are mapped onto the attributes that locate a colour within a particular colour model. A string-valued attribute *colourModel* (read only) is defined in the *Colour* object type to name the colour model in which that colour is defined. The use of a given colour model imposes an interpretation on the dimensionality of a colour object and on each of the components of that object. To support such interpretations, a number of integer constants are defined to identify the component of a colour object that is used to store particular characteristics of a colour:

Constant	Value	Description
ColourRGBA	1	Red component of RGB
ColourRGBG	2	Green component of RGB
ColourRGBB	3	Blue component of RGB