Centrum voor Wiskunde en Informatica

REPORT*RAPPORT*

PREMO: an emerging standard for multimedia presentation

I.Herman, G.J. Reynolds and J. Van Loo

Computer Science/Department of Interactive Systems

CS-R9554 1995

Report CS-R9554
ISSN 0169-118X

CWI
P.O. Box 94079
1090 GB Amsterdam
The Netherlands

PREMO: An Emerging Standard for Multimedia Presentation

Ivan Herman, Graham J. Reynolds

CWI
P.O. Box 94079, 1090 GB Amsterdam, The Netherlands

Email: {Ivan.Herman,Graham.Reynolds@cwi.nl}

James van Loo

SunSoft, mTV 10–228
2550 Garcia Avenue, Mountain View, CA 94043–1100, USA

Email: james.vanloo@sun.com

Abstract

ISO/IEC JTC1/SC24 are developing a standard for the presentation of multimedia objects, called PREMO (Presentation Environments for Multimedia Objects). PREMO is aimed at application developers who want to include multimedia effects into the applications, but do not want to restrict themselves to model of multimedia documents, which is prevalent in multimedia applications today. This report gives an overview of the current status of PREMO.

AMS Subject Classification (1991): 68N15
CR Subject Classification (1991): D.1.5, D.2.0, D.2.6, H.5.1, I.3.2, I.3.6, K.1
Keywords and Phrases: PREMO, multimedia, object models, active objects, standards
Note: This paper will be published in the IEEE Multimedia journal

1. Introduction

Multimedia is a booming industry; this statement has become commonplace these days. There are a growing number of multimedia tools and multimedia environments available on all major computing platforms, and multimedia products have begun to appear in the end consumer market, too.

However, this enormous expansion of multimedia, and the expectations raised by it, tends to hide a very significant fact, namely that virtually all multimedia environments available today are based on a relatively restrictive *multimedia/hypermedia document* paradigm. This paradigm regards the concept of multimedia documents as an extension to traditional textual documents, by adding other forms of non–textual information such as video sequences, sound tracks, still and animated images. By adding active links, multimedia documents become hyper-

media documents. The multimedia market is dominated by this paradigm, which has even imposed its own terminology onto the entire field (one speaks of “authoring systems”, “multimedia titles”, etc.)

Obviously, the success of this paradigm is due to the fact that it is easy to understand, even for naive users, and to encompass a large number of applications; it is not our purpose to criticize it here. On the other hand, one should not be blinkered by this paradigm either; there are a number of important application domains where it essentially breaks down. Applications where parts of, or indeed all of, the multimedia information is synthetically generated at run–time, perhaps based on some internal condition of the application, or where different media specific information are merged, transformed and transmuted into one another, either interactively or as a result of a complex calculation, do not fit easily or elegantly into the model of multimedia/hypermedia documents. This class of applications includes high–end simulation environments, systems with advanced multi–modal user interfaces, scientific and engineering visualisation systems (possibly bound to high–performance computing environments), geographic information systems, and virtual reality. All these application areas will require the use of various media in the future. As an example, contemporary virtual reality systems are based primarily on advanced synthetic 3D graphics and audio localization techniques, but in the future they will clearly include photorealistic rendering (e.g., radiosity and ray–tracing), advanced sound rendering, video, and other more esoteric effects. The necessity for the synergy between multimedia and for example, virtual reality, is recognized today but this is still largely an open area for research and development.

Today’s application developers needing to realize high–level multimedia applications are essentially left on their own. There are only a few programming tools which allow an application developers to create multimedia effects based on a more general model than multimedia documents. The most representative systems are Kaleida Lab’s ScriptX, Sybase’s GainMomentum, the World Toolkit for virtual reality systems, or the MADE toolkit commercialized by Groupe Bull, France[1]. In any case, there is currently no available ISO standard encompassing these needs. A standard in this area should focus on the *presentation* aspects of multimedia, and much less on the coding, transfer, or hypermedia document aspects, which are covered by a number of other ISO or de–facto standards. It should also concentrate on the *programming* tool side, and less on the (multimedia) document format side. These are exactly the main concerns of the PREMO Standard, which is the subject of this paper.

It is quite natural that the initiative for a standardization activity aiming at such a specification came from the group which has traditionally concentrated on presentation aspects in the past 15 years, namely ISO/IEC JTC1/SC24. Indeed, this is the ISO subcommittee whose charter has been the development of computer graphics and image processing standards in the past. The Graphical Kernel System[9] was the first standard for computer graphics published by this subcommittee of ISO; it was followed by a series of complementary standards, addressing different areas of computer graphics. Perhaps the best known of these are PHIGS[2], PHIGS PLUS[3], and CGM[4]. The subcommittee has now turned its attention to presentation media in general as a way of augmenting traditional graphics applications with, e.g., audio, video, or still images facilities in an integrated manner. The need for a new generation of standards for computer graphics has already emerged in the past 4–5 years to answer the challenges raised by new graphics techniques and programming environments; it is extremely fortunate that the review process to develop this new generation of presentation environments coincided with the emergence of multimedia. Hence, a very fruitful synergy effect can be capitalized on.

Notwithstanding important differences in their functionality, all standards cited above share a common architectural approach, which, although not a requirement defined within the documents, has resulted in implementations that are large monolithic libraries of a set of functions with precisely defined semantics. They reflect an approach towards graphical software libraries predominant in the seventies and the eighties. These standards have little chance of providing appropriate responses to the rapid changes in today’s technology, and in particular, they fail to fit into the software and hardware system architectures prevailing on today’s systems. The JTC1 SC24 subcommittee recognized the need to develop a new line of graphics standards, along radically different lines from previous approaches. It also recognized that any new presentation environment should include more general multimedia effects to encompass the needs of various application areas. To this end, a new project was started at an SC24 meeting at Chiemsee, Germany, in October 1992; subsequent meetings resulted in a draft for a new standard called PREMO (Presentation Environment for Multimedia Objects). This new work was approved by ISO/IEC JTC1 in February 1994, and is now a major ongoing activity in ISO/IEC JTC1 SC24 WG6.

The major features of PREMO can be briefly summarized as follows; section below gives a more detailed overview of the main PREMO features.

- **PREMO is a *Presentation Environment***. PREMO, as well as the SC24 standards cited above, aims at providing a standard "programming" environment in a very general sense. The aim is to offer a standardized, hence conceptually portable, development environment that helps to promote portable multimedia applications. PREMO concentrates on "presentation techniques"; this is what primarily differentiates it from other multimedia standardization projects.
- **PREMO aims at a *Multimedia* presentation**, whereas earlier SC24 standards concentrated either on synthetic graphics or image processing systems. Multimedia is considered here in a very general sense; high–level virtual reality environments, which mix real–time 3D rendering techniques with sound, video, or even tactile feedback, and their effects, are also within the scope of PREMO.
- **PREMO is *Object Oriented***. This means that, through standard object–oriented techniques, a PREMO implementation becomes extensible and configurable. Object–oriented technology also provides a framework to describe distribution in a consistent manner.

The specification of the PREMO standard is an ongoing activity and the goal is to reach the stage of a Draft International Standard in 1998.

2. An overview of PREMO

From an editorial point of view, PREMO is a multipart standard. This means that the full PREMO standard consists of a number of Parts (currently 4), which, although there are interdependencies, progress through the usual ISO working scheme independently of one another. It is to be expected that new parts of PREMO will also be added to the PREMO standard in the years to come.

Structurally, PREMO is divided into *components*. A component in PREMO is a collection of object types and non–object data types, from which objects and non–objects can be instantiated. Objects within one component are designed for close cooperation and offer a well–defined set of functional capabilities for use by other objects external to the component. A component can offer *services* usable in a distributed environment, or they may be used as a set of objects directly linked to an application. Components may be organized in component inheritance hierarchies. At present, all PREMO Parts, except for the first one, describe a specific component.

A PREMO compliant implementation is not mandated to implement all PREMO components; it may choose to implement only some of them, and rely on the services of other components through inter–object communication. Conformance rules under development will give clear guidelines as to what a PREMO compliant implementation should be. It is, in any case, mandated that a compliant implementation shall implement full components, and not parts of them only.

At present, four PREMO parts are registered as official ISO work items. These are:

- **Part 1: Fundamentals of PREMO**[5]**.** This Part contains a motivational overview of PREMO giving its scope, justification, and an explanation of key concepts. It describes the overall architecture of PREMO and specifies the common semantics for defining the externally visible characteristics of PREMO objects in an implementation-independent way. See section for more details.
- **Part 2: Foundation component**[6]. This component lists an initial set of object types and non–object types, useful for the construction of, presentation of, and interaction with multimedia information. Any conforming PREMO implementation shall support these object types. These types include fairly traditional types, such as event handlers, aggregates, as well as types more specifically tailored to the needs of interactive multimedia systems.
- **Part 3: Modelling, Presentation, and Interaction Component**[7]. This component combines media control with modelling and geometry. This is an abstract component from which concrete modelling and presentation components are expected to be derived. Thus, for example, a virtual reality component that is derived, at least

in part, from this component, might refine the renderer objects defined in Part 3 to objects most appropriate in the virtual reality domain.

- **Part 4: Multimedia System Services**[8]. This component provides an infrastructure for building multimedia computing platforms that support interactive multimedia applications dealing with synchronized, time–based, media in a heterogeneous distributed environment.

  Whereas Parts 1, 2, and 3 have been developed entirely within the PREMO Rapporteur Group, the origin of Part 4 is different. The specification for the Multimedia System Services was originally developed in the form of a "Recommended Practice" document of IMA[1], and has undergone an internal refereeing process within IMA. IMA has recently submitted the final version of the Multimedia System Services to ISO as Part 4 of PREMO (this will also necessitate the adaptation of the document to abide by the rest of the PREMO framework).

The current PREMO activities also include the definition of three new components, although at the time of writing this paper, these components are not yet officially registered. Consequently, the exact title and numbering of these components may change in the future. These are:

- **Part 5: GKS-94 Component**. The original GKS standard has undergone a review[10] which resulted in the publication of a new version of the GKS standard in November 1994. The goal of this component is to define a compatible, but object–oriented, specification of the revised standard.
- **Part 6: PHIGS PLUS Component**. Like the GKS-94 component, this component aims at a compatible, but object–oriented, specification of the widely used PHIGS PLUS standard.
- **Part 7: Multimedia Virtual Environments Component**. In contrast to Parts 5 and 6, this component will result in a completely new specification to provide a standard base for high–end virtual reality applications.

Finally, components for 3D Audio facilities, as well as for the realization of an MHEG engine, are also in planning

The following sections give brief overview of each of the four existing PREMO components, although, given the limitations imposed by the size of this paper, a number of details have necessarily been omitted[2].

2.1. PREMO Part 1: Fundamentals of PREMO

At the earliest stages of the PREMO project specification it became clear that a concise framework would be needed to ensure the smooth cooperation among objects within PREMO and to provide a consistent approach to some of the technical issues raised by multimedia programming in general. This framework took the form of a precisely defined object model. This object model constitutes the major technical topic of this Part of PREMO. The rest of the Part describes the scope and purposes of the PREMO standard in general.

The object model is traditionally based on the concepts of subtyping and inheritance. It is also very pragmatic in the sense that it includes, for efficiency reasons, the notion of non–object (data) types, as it is done in a number of object–oriented languages like C++, and in contrast to "pure" object–oriented models such as SmallTalk. The PREMO object model originates from the object model developed by the OMG consortium for distributed objects, where some aspects of the OMG model have been adapted to the needs of PREMO. However, it is the goal of the PREMO object model to ensure a compatibility with the OMG specifications, i.e., PREMO implementations should be realisable using the implementations of distributed object services based on OMG's CORBA and COSS specifications.

---

1. IMA (Interactive Multimedia Association) is a major industrial consortium regrouping hardware vendors like Sun, Digital Equipment, HP, IBM, Kaleida Labs, Siemens, and others.

2. The interested reader may also refer to the PREMO document itself, which is publicly available, as well as the various ISO documents and other publications. The World Wide Web site "`http://www.cwi.nl/Premo/`" gives a good starting point to navigate through and access all available documents.

In PREMO, a strong emphasis is placed on the ability of objects to be active. This feature of PREMO stems primarily from the need for synchronisation in multimedia environments. Conceptually, different media (e.g., a video sequence and a corresponding sound track) may be considered as parallel activities that have to reach specific milestones at distinct and possibly user definable synchronisation points. In many cases, specific media types may be directly supported in hardware. In some cases, using strictly specified synchronisation schemes, the underlying hardware can take care of synchronisation. However, a general object model should offer the capability of describing synchronization in hardware independent terms as well (see also other reports[11,12,13] for similar approaches taken in multimedia programming systems).

Allowing objects to be active does not contradict the OMG object model. However, some details of object requests have to be specified in more precise terms. In PREMO, objects may define their operations as being *synchronous*, *asynchronous*, or *sampled*. The intuitive meaning of these notions is:

- If the operation is defined to be *synchronous*, the caller is suspended until the callee has serviced the request.
- If the operation is defined to be *asynchronous*, the caller is not suspended, and the service requests are queued on the callee's side. No return value is allowed in this case.
- If the operation is defined to be *sampled*, the caller is not suspended, but the service requests are not queued on the callee's side. Instead, the respective requests will overwrite one another as long as the callee has not serviced the request.

The only unusual feature of this model, compared to traditional message passing protocols, is the introduction of sampled messages. Yet, this feature is not unusual in computer graphics. Consider the well–known idea of sampling a logical input device to obtain locator position values. A separate object that models a locator can send thousands of motion notification messages to a receiver object, and the latter can simply "sample" these messages using the sampled message facility.

Synchronization is modelled using the notion of synchronization of concurrent processes, i.e., active objects in PREMO. The model provides a clean and straightforward framework on which complex synchronisation facilities can be built.

PREMO relies on an external environment to create and destroy objects and object references; although it has some simple requirements on these external facilities, it does not specify all the details. Consequently, services like the object lifecycle services of OMG[14], or the creation and destruction facilities defined in various object–oriented languages are directly usable when implementing a full PREMO environment.

Part 1 of PREMO also introduces the notion of service objects and components, which give a way to structure PREMO implementations. A component in PREMO is a set of related object types and non–object types that comply with the PREMO object model and from which objects may be instantiated. A component offers a set of services and may also require services from other components.

In addition to the set of object types and non–object types included in a component, a component also defines a configuration specification that makes explicit all dependencies between object types and their operations within the component and any dependencies it has on other PREMO components. The configuration specification describes these dependencies as follows:

1) A component ***A*** may depend on itself if there are object types in ***A*** that are either:
   i) derived from other object types defined within ***A*** (type dependency), or
   ii) whose behaviour depends on operations defined by object types in ***A*** (behavioural dependency).
2) A component ***A*** may depend on another component ***B*** if their are object types in ***A*** that are either:
   i) derived from other object types defined within ***B*** (type dependency), or

ii) whose behaviour depends on services provided by object types defined within ***B*** (behavioural dependency).

The various possible dependencies are non–exclusive; a component may have internal and external dependencies that may be in terms of both type and behavioural dependency.

The configuration specification of a PREMO component will make provision for PREMO implementations to offer automatic configuring mechanisms. Such mechanisms may allow for an implementation of a component to interoperate with other component implementations.

2.2. PREMO Part 2: Foundation Component

The foundation component is a collection of *foundation objects*. Foundation objects are those which support a fundamental set of services suitable for use by a wide variety of other components.

It is beyond the scope of this paper to give an exhaustive specification of all foundation objects defined in the foundation component; only some highlights are given here.

2.2.1 Fundamental Object Behaviour

This notion is covered by the specification of a so–called `PREMOObject` object type, which is the common supertype for all PREMO object types. By virtue of subtyping, all operations defined on this object type are available for all other PREMO Objects. These operations include initialisation and destruction operations, inquiry–type operations: an object can return information on its type, on its supertypes, etc., and, finally, a set of operation to manage dynamic properties (i.e., key–value pairs which can be dynamically assigned and inquired for an object).

It is interesting to note that a number of object–oriented systems do not offer such inquiry facilities by default, although interactive applications have a clear need for them. Indeed, these inquires play a fundamental role in the dynamic adaptation of a system to the set of and the quality of available resources, thereby making application programs inherently more portable. The example of OpenInventor[15], which defines these inquires within its own object model, is characteristic in this respect.

2.2.2 Enhanced Data Objects

The semantics associated with a data object (abbreviated as `EDO`-s) define the construction and modification interface of a particular data object. Examples are geometric 2D or 3D points, colour, matrices, with related operations and other attributes, video frames, frequency spectra, etc.

2.2.3 Event Handler Objects

This object type offers the necessary operations for the implementation of event management within PREMO. Event management plays a fundamental role within interactive systems in general, and the "*Event Model*" of PREMO aims at providing this basic mechanism.

The essential feature of the event model is the separation between the source of the events (e.g., a mouse, some external hardware, or other PREMO objects implementing a more complex interaction scheme), and the recipient of these events. Sources broadcast the events without having any knowledge of which objects would receive them; this is done by forwarding the event instance to special PREMO `EventHandler` objects. Prospective recipients of events register with these `EventHandler` objects, placing a request based on the event type and optionally other more complex constraint specifications. The recipients are then notified by the `EventHandler` on the arrival of an event, together with information on the source of the event. This simple mechanism constitutes one of the main building blocks for the creation of more complex interaction patterns in PREMO.

The standard also defines two important subtypes of event handlers, namely so–called synchronization points and 'AND' synchronization points. The additional feature added to these types, with respect to general event handler

objects, is a possibility to constrain the events which can be dispatched, and to possibly delay the notification of event recipients.

2.2.4 Controller Objects

The role of a `Controller` is to coordinate cooperation among objects. A `Controller` object is an autonomous and programmable finite state machine (FSM). Transitions are triggered by messages sent by other objects. Actions of the FSM correspond to messages sent to other objects. The actions of a `Controller` object may cause messages to be sent to other `Controller` objects, thus a hierarchy of `Controllers` can be defined.

2.2.5 Clock Objects

These objects provide a unified interface to the system's view of real–time clock. The clock object type assumes the existence of two non–object types: `Time`, to measure elapsed ticks (realized, for example, as a 64 bits integer), and `TimeUnit`, which (as an enumerated type) defines the unit represented by each clock tick, for example an hour or a micro–second. Specifically, the clock object type supports an operation, `inquireTick`, that returns the number of ticks that have occurred since the start of era defined for PREMO[1]. However, the accuracy in various units with which particular PREMO implementation can describe the elapsed duration since the start of era will vary, and for this reason the clock object type defines a read–only attribute for determining the performance of the local object.

2.2.6 Synchronization Objects

The synchronization facilities included in PREMO are based on an event–based synchronization model[16]. In this model, each synchronizable object is considered to progress autonomously along an internal, one dimensional coordinate space, which may be integer, real, or time–based (e.g., an integer coordinate may be used to describe frames in a video sequence). may have a set of reference points (e.g., video frames). Points on this space are referred to as *reference points*; for each reference point, an object and an operation can be registered, together with an event instance (the synchronization event). This operation is invoked by the synchronizable object when a reference point is crossed, using the event stored at the reference point as an argument. A reference point may also contain a boolean flag, which may instruct the synchronizable object to suspend itself and wait for an external message request to continue its progress. Typically, a reference point would refer to an event handler object or a controller object. For example, in the first case, the synchronization event may be dispatched to a number of event recipients (either unconstrained or, through the usage of the synchronization points described above, through some additional constraints), thereby creating complex synchronization patterns among various synchronizable objects.

The synchronizable objects serve as a common supertype for a number media–specific types, like video, audio, animated graphics, etc. These types are defined in the various components of PREMO.

The event based synchronization model can be used for synchronization patterns where traditional, purely time–based synchronization is not sophisticated enough. A purely time based synchronization can also be built on top of this model, and this is done by combining (through multiple inheritance) the purely event based synchronization objects with the clock object. The resulting object type, called `TimeSynchronizable` in the document, has an additional attribute, called `speed`, which characterizes the pace of progression of the object in time.

2.2.7 Aggregate Objects

This is a class of object types within PREMO, offering a range of traditional aggregation facilities, like lists, arrays, tables, etc. Although there are a number of aggregate object specifications and/or implementations available on various systems and programming environments, there is no one which would be either a de–jure or a de–facto standard, which PREMO could rely on. On the other hand, the possibility for complex aggregation facilities

1. This start of era is currently defined for all PREMO systems to be 00:00am, 1st January 1995, UTC; the exact definition may change in the future.

plays a fundamental role in interactive presentation systems; to have a clear common understanding within PREMO, it was necessary to include these object type specifications into the standard.

2.2.8 Producer and Porter Objects

`Producer` objects provide an encapsulation for defining the processing of `EDO` objects and the production of refined or transmuted `Data` objects. Producer objects may receive `Data` objects from any number of sources and deliver `Data` objects to any number of destinations. Specific subtypes of the `Producer` type may place restrictions on the number of sources and destinations of `Data` objects if necessary, on their types, etc. Specific types of `Producer` object are characterized by the behaviour made visible through their associated sets of operations. An example of a `Producer` object is an object which receives information about a polyline and creates an array of integer vectors, representing pixels, which may be further processed by another producer object to display the pixels on a display surface.

A `Porter` object is the PREMO foundation object which interconnects to systems and environments defined outside of PREMO, e.g., files, physical devices. A porter object can be used to export the output of PREMO (an `OutputPorter` object) or to import input into PREMO (an `InputPorter` object). Subtypes of this object provide consistent "virtual device" interfaces to specific external devices, for example, the frame buffer of the display, a clock, a microphone, a speaker, or a frame grabber. In addition, other derived types may provide the basis for database enquiry, with further derived types for specific database systems. `Porter` object may also be connected to `EventHandler` objects (see section) as possible event sources; some of the external resources may indeed be completely or partly event–driven (e.g., buttons of a mouse).

Both `Producer` and `Porter` objects may be used to create networks of cooperating processing nodes. For example, subtypes of `Producers` may be used as renderers, which operate on geometric objects, and which forward data to other renderers to form a rendering pipeline. A `Porter` object may be placed at the end of such a pipeline to produce, e.g., some visual or audio effect.

Each node of such processing networks can operate in so–called data–driven or control–driven mode. It is therefore possible to set up fully data–driven networks (also referred to as "data–flow", or "push model" networks) or fully control–driven network ("control–flow" or "pull model" networks). In the former case, the operation of each node is triggered by the arrival of new data, and the produced data is automatically forwarded to the target nodes, whereas, in the latter case, each node takes the initiative of requesting data from possible sources and stores the produced data locally (until another node requests them). In fact, because each node's operation mode and, furthermore, the operation mode of both the input and the output "side" of a node can be can be set individually and independently of one another, very complex processing patterns can be set up using these fundamental building blocks.

2.2.9 Other foundation object types

The object types listed earlier in this section are those which are already part of the current PREMO document. However, PREMO is still an evolving specification, and there are a number of discussion going on within the subcommittee which may result in the introduction of new objects and/or the modification of the current ones.

2.3. PREMO Part 3: Modelling, Rendering, and Interaction Component

The goal of this component is to define a family of object types that describe modelling, rendering, and interaction in abstract terms. Part 3 is considered to be an abstract component in the sense that concrete presentation functionalities are intended to be realized in further, more specialized, components which make use of the concepts and object types defined in Part 3.

The model for presentation, as described in Part 3, originates from an existing SC24 standard called the Computer Graphics Reference Model (CGRM)[17] (see also the paper on an introduction to CGRM[18]). Although originally developed for computer graphics, this reference model can been adapted for the purposes of multimedia presentation; in this sense, Part 3 can be regarded as casting the general notions of the CGRM into the framework of the PREMO object model and the set of object types defined in Part 2.

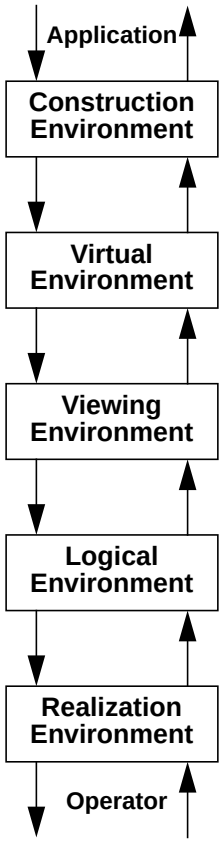

Figure 1. Environments in Part 3

The key idea of Part 3 is that multimedia presentation can be considered as a series of transformation steps between the application and the operator. Transformation is considered in a very general sense, e.g., it includes the transformations among different colour models, coding and decoding algorithms, etc. Transformation steps are modelled in PREMO in terms of five abstract levels called *environments*: construction, virtual, viewing, logical, and realization (see Figure 1). These environments form a processing network, using the object types developed in Part 2. Information, e.g., multimedia data, attributes, input requests and data, results of inquiries, etc., can flow in both directions between two adjacent environments. Although Figure 1 shows a simple, pipeline–oriented network, this is only the basic case. More generally, any number of fan–ins and fan–outs are possible between two adjacent environments. For example, it is possible for an instance of a virtual environment to be connected to several instances of viewing environments, thereby resulting in multiple views of the same model.

On the output side, the role of each type of environment can be summarized as follows.

- *Construction environment*: In this environment, the application data to be displayed is prepared as a model from which specific presentation scenes may be produced. The application may only edit the model in the construction environment.
- *Virtual environment*: In this environment, a scene of the model is produced. The scene consists of a set of virtual output primitives ready for presentation. The geometry of these virtual primitives is completely defined in all dimensions so that scenes are geometrically complete.
- *Viewing environment*: In this environment, a picture of the scene is generated by projection.The picture consists of a set of viewing output primitives ready for completion. Output primitives in the viewing environment may have a lower geometric dimensionality than in the virtual environment.

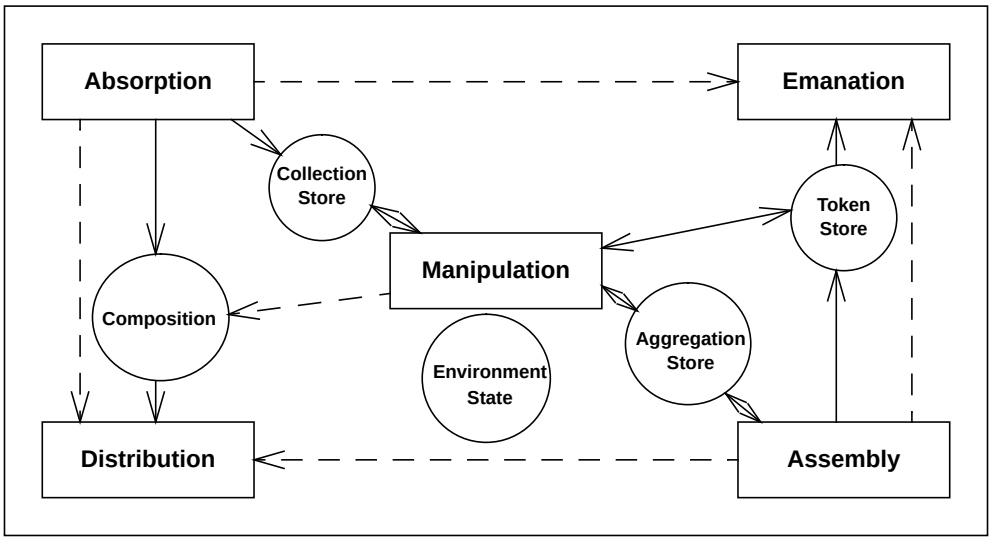

Figure 2. A PREMO Environment

- *Logical environment*: In this environment, a presentable version of the picture is completed ready for realization. The presentable image consists of a set of logical output primitives. Associated with each output primitive is a set of properties associated with completion.
- *Realization environment*: In this environment, a display of the presentable picture is presented. The display consists of a set or realization output primitives. This display need not necessarily correspond to a physical display, it can also be a sound output device, a video hardware, or a logical driver.

The symmetry between input and output is reflected on the diagram; as for output primitives, Part 3 makes a series of statements on the role of each environment in the processing of input tokens.

The internal model of each environment follows the same structure; it is therefore possible to describe an environment through a general PREMO object type which is then specialized for the various types of environments described above. Internally, an environment consists of a network of processing objects, cooperating with various types of aggregates which store multimedia data (or their references) locally. Figure 2 gives an overview of the general structure of an environment: the rectangles represent subtypes of PREMO `Producer` objects (see page 6), specialized for the needs of the environments, whereas the circles represent various types of aggregates. Dashed and continuous arrows represent control and data flows, respectively, among the different entities.

The PREMO object types defining the various environments are not the only types defined in Part 3. Both output primitives and input tokens are defined in terms of a large palette of general PREMO types. The properties of output primitives specify their geometry and appearance. These properties are currently classified as spatial, visual, aural, tactile, textual, and identification properties, although the exact details of this classification is still the subject of further debate.

2.4. PREMO Part 4: Multimedia System Services

Any presentation system for graphics and multimedia faces an identical problem: the demand for generality and portability often conflicts with the large variety of available presentation hardware which all follow their own specification. The inclusion of multimedia into modern presentation systems made the situation even worse: not

only did the variety of hardware elements increase, but the various forms of coding and compression techniques used in video, audio, and image processing have added to the confusion.

The goal of Part 4, i.e., the Multimedia System Services (MSS), is to alleviate this problem, and to hide device dependencies from other components of PREMO. The Multimedia System Services component defines itself as a "middleware", i.e., a system software component lying in the region between the generic operating system and specific applications, including such higher level components as Part 3 of PREMO. As middleware, MSS marshals lower–level system resources to the task of supporting multimedia processing. MSS does not contain any notion of geometry, of presentation models, of higher level composition of abstract presentation entities; it concentrates on the lower level management of multimedia devices, their connections, distributions, resources, etc. Consequently, Part 4 plays a complementary role to the one played by Part 3, and the force of PREMO results from the cooperation of these two entities.

The objects which comprise the framework of MSS describe a data –flow graph[1]. The graph nodes, known as *device* objects, provide an abstraction for media execution. The graph arcs, known as *connection* objects, provide an abstraction for stream transport. The clear separation of functions allows some organizations (e.g., software or hardware vendors) to contribute the media aware objects, while other organizations contribute the transport aware objects. Figure 3 illustrates the concept. The video capture device produces a video stream for the video compression device. The filter device receives streams from both the audio capture device and the video compression device. The filter device interleaves the streams to produce a movie stream. The device objects on the receive side reverse the execution sequence. It is valid, of course, to package these atomic devices into a composite device. A movie capture device, for example, would produce a movie stream, but not expose what is inside the device.

The `device` type and the `connection` type inherit from the `resource` type. The resource type allows a client to describe its expectations about the quality of the media stream. The quality metrics, for example bandwidth bounds, apply to both device objects and connection objects.

The device objects transmit and receive the media stream through a device *port*. The interface to the device object provides methods through which a client can discover the available ports. A device object which imports a stream is known as a display device; a device object which export a stream is known as a capture device; a device object which both imports and exports a stream is known as a filter device. While the device name often implies the ports, the client can always discover the configuration from the device object itself, as one attribute of each port is the direction

The interface of the device type provides methods to discover the `format` objects and `protocol` object available at each port. The motivation for the format object is interoperation between remote device objects. If independent organizations can provide device objects, how does a client know that the media format which one device object produces, versus the media format which another device object consumes, are compatible? The solution requires that the device object describe the Format objects which are feasible at each device port.

The `Format` object is one example of an object which is the subtype of `KeyValue` type. The `KeyValue` type provides operations to `compare`, `constrain`, and `select` values within a key, value space. With the `compare` operation the client provides keys and values of interest to the client. The `KeyValue` object intersects the client space with the values which the object can realize. If the space intersect, the object reports the intersection. If the space fails to intersect, the object reports the values which it found offensive. The `compare` operation does not change the object state. To affect the values which the object is to realize, the client invokes the `constrain` operation. This operation instructs the object to consider only the intersection of the format space to be feasible. If there are still options available, the client invokes the `select` function. The function informs the object that it is to select a value for those attributes for which multiple values are available.

The client can associate multiple format objects with each device port. If the client knows that the session involves multiple sections with distinct formats, the client would associate multiple format objects with the device

1. While the metaphor is a data–flow graph, the execution technique of the objects which comprise the data–flow graph can be control–flow.

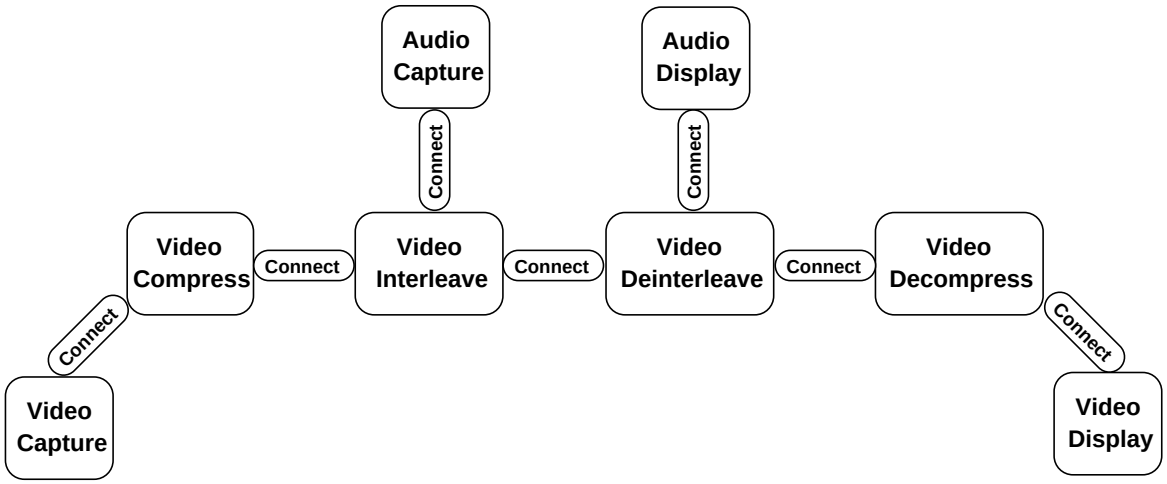

Figure 3. Multimedia Data Flow Graph

ports. The client would then attach the proper format object in response of the advance of the stream time or in response to stream state transitions.

Just as the `Format` type describes the media context, the `Profile` type describes the transport context. The motivation is again interoperation. The `Profile` type is a subtype of the `KeyValue` type, but augments the interface with keys and values which are specific to transport.

The connection object abstracts the transport mechanism. The object provides operations to `attach` and `detach` a device port to a media stream. The semantics of the `attach` operation are that the connection object will report an exception if the format objects and transport objects bound to the device port can not interoperate with the media stream. If interoperation is feasible, the connection object converges the format context and protocol context to compatible values. The feature allows a caller to defer the configuration phase to the `Connection` object. There is no restriction that the context of the device port which the caller provides in the `attach` operation matches the device ports already bound to the stream. If a proactive connection detects format conflicts, for example, it could interpose a device object which transcodes the format.

To control the advance of the media stream, the framework provides a `Stream` object type. The base interface provides operations to `pause(T)` and `resume(T)` the stream as a function of stream time. The caller binds the `Stream` object to a specific location in the data flow graph. The object to which the stream is bound cascades the control functions to the adjacent objects which share the stream. Since the stream object requires a state machine, the type inherits from a state machine type. The basic state machine type allows a client to detect state transitions. The stream type extends the basic type with the state specific to stream control. One application of this type is the situation where the session involves multiple passages and each passage relates to a distinct device object. A caller can leverage the interface to detect a transition at one device, for example the transition to a pause state, and then invoke an operation, for example `resume`, on another device.

The basic stream type is often adequate for the situation where all the data flow objects already receive a precise clock. The caller can also explicitly `attach` a time source to certain stream objects. The caller provides a transform to relate the time coordinate space of the master to the time coordinate space of the slave. While the master is often a clock object, any object which reports time can be the master. The caller could attach a time stream of an audio stream to a stream object of a video device, for example. The video stream, in this example, synchronizes to the audio stream.

3. Formal Specification of PREMO

The graphics standards community have in the past employed formal methods in only a very limited sense. The semantics of first generation graphics standards, such as GKS and PHIGS, were described using natural language, and in some cases this has meant that ambiguities have crept into the specifications. The PREMO group of ISO/IEC JTC1/SC24 plans to address this problem by employing formal methods at an early stage and to continue this activity throughout PREMO's development. The intention is to provide a formal specification of the PREMO object model and some of its components, where the main emphasis is placed on feeding results back into the standard's development. This is essentially a complimentary activity and it is not currently planned that this should replace the usual natural language description.

The formalism adopted for the formal specification of PREMO is based on the Z and the Object–Z languages. The choice of Z[19,20] and Object–Z[21] for the work described here followed from recommendations of the SC24 Study Group on formal description techniques[22] and was motivated by three considerations.

1) The object–oriented nature of PREMO functionality favours the use of an object–oriented formal description technique; this is offered by Object–Z.
2) Within ISO/IEC, the only formal description technique which has the status of International Standard is LOTOS[23]. LOTOS is a language based on process algebra. The present PREMO work is concerned with the description of the PREMO object model, including intra and inter object communication. LOTOS was considered inappropriate for this work, at least initially, because it is not state–based and not object–oriented. However, some work is planned to look in more detail at the appropriateness of LOTOS for describing the communications aspects of PREMO.
3) There is considerable expertise in the group in both Z and Object–Z, more so than in process based notations. The existence of appropriate expertise in formal description techniques is a significant hurdle to overcome in gaining acceptance for formalism. It is important that experts in a standardization committee who do not have expertise in writing formal descriptions should at least have the opportunity to learn to read them at relatively low cost. This implies that there should be good access to training materials such as books, courses and case studies. There is excellent material on Z available under all 3 categories.

It is obviously not possible, within the framework of this paper, to enter into the details of the formal specification of PREMO. The interested reader may consult a separate paper[24] which gives a good overview of the methods employed, and of the first results of this activity.

4. Further Work

The PREMO standardization work is an ongoing activity and inevitable the main documents still undergo significant change. Some general problems, which may have profound influences on various parts of the document, are still the subject of discussions among the participating experts. This includes:

- Inclusion of constraint–management techniques into PREMO. Geometric and other types of constraints constitute an extremely important set of tools for a large number of complex interactive applications, including animation, simulation systems, etc. It is, however, notoriously difficult to combine the demands of constraint management that are generally declarative with the procedural nature of object–oriented programming paradigms. Possible inclusion of constraint management techniques into PREMO may therefore modify the current object model, as well as the behaviour of the existing foundation objects.
- Inclusion of 3D sound rendering techniques into a virtual reality environment is obviously of a great importance[25]. However, this technique is still relatively new, and it is not yet clear whether modifications are necessary on the current type specifications of Part 3 to encompass the needs of realistic sound rendering.

5. An example of multimedia integration

The objects and functionality discussed so far essentially define a framework in which sophisticated multimedia applications can be developed. To see just how this might be achieved in PREMO we can consider a prototypical example application which contains elements of modelling, synchronization and interaction that are often required in complex multimedia environments. Our goal with this example is simply to give an outline of how PREMO's object types can be applied. Clearly, since much of PREMO is currently undergoing technical review, this example should not be taken as the "standard" way to realize this kind of application.

The example application is a model of a room. The room contains a chair, a TV set, and a person (called the viewer). The operator (i.e., the end user of the application) is able to interact with the room in a limited number of ways. He may move the chair, the TV set and the viewer, and he may change the view of the room. The operator can also interact with the TV set by pressing an on/off button. A TV programme, consisting of a video with accompanying audio and subtitles, is played on the TV set when it is switched on. An important aspect of this example is the fully integrated nature of the video and audio components of the TV programme's presentation. The video component may have a direct impact on the rendering and lighting calculations involved in presenting the room. Similarly, the rendition of the audio component may be affected by the location of the TV set within the room and other properties, such as the viewer's position.

The presentation aspects of the example are illustrated by the hierarchy shown in Figure 4.

There are two kinds of objects used within this hierarchy. The viewer, chair, case, on/off button, audio, video, and subtitles are all objects that have some form of perceivable presentation, either visual or aural. In effect, these objects are media objects. We do not concern ourselves with their complexity in this paper, except to remark that they include behaviour that allows them to synchronize with other objects in the PREMO system. The remaining

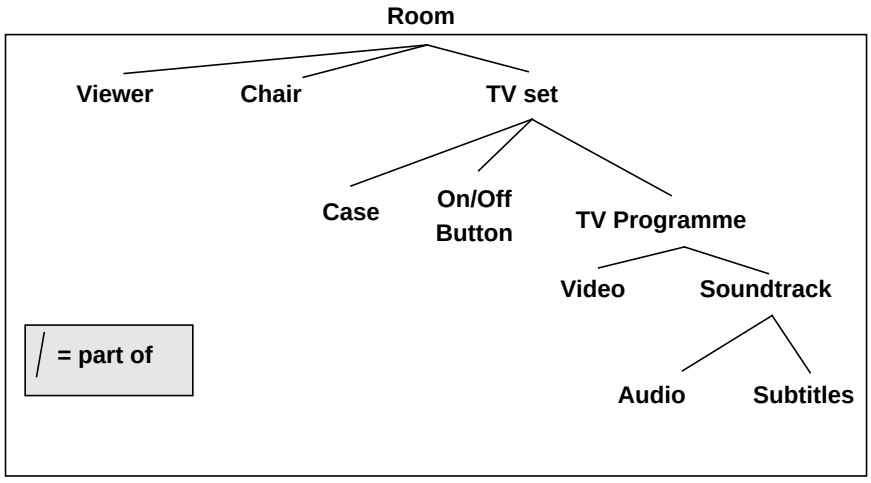

Figure 4. The example 'part–of' hierarchy

objects, TV set, TV programme and soundtrack, are subtypes of synchronizable object — one of the object types defined in the foundation component[1]. In addition to their synchronizable behaviour, they provide an ability to manage their children. In effect, they are composite objects.

This object hierarchy forms the *composition* in one or more of the Part 3 CGRM environments. To change the view of the room would require interaction with the composition in the viewing environment. To move the

1. As mentioned earlier, these synchronization objects have been proposed for inclusion in the foundation component.

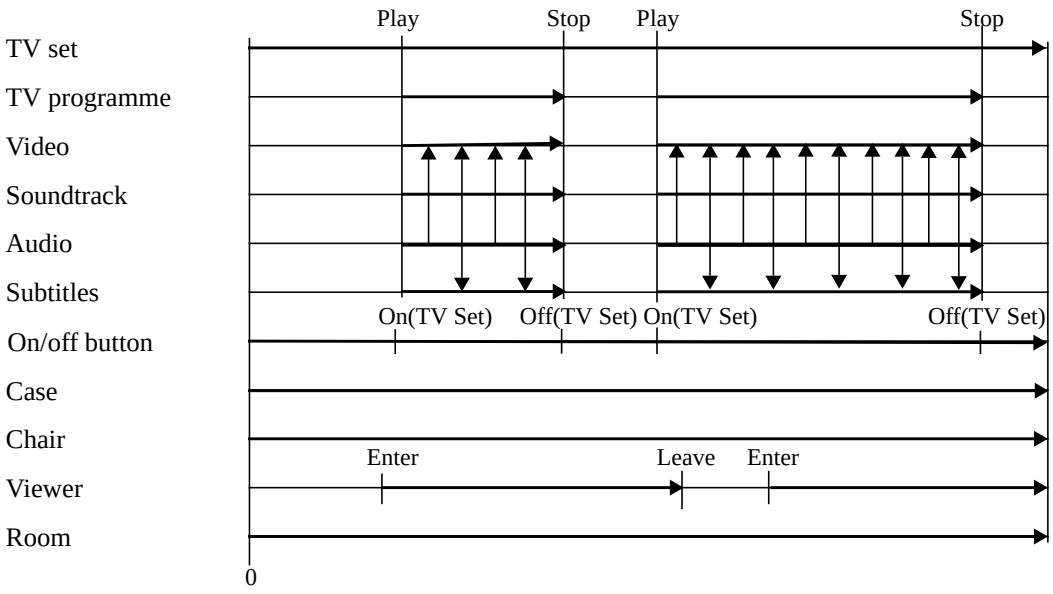

Figure 5. Timing diagram for room example

viewer, chair, or TV set within the room requires interaction with the composition in the virtual environment. To add extra furniture to the room requires interaction in the construction environment. Starting the display of the TV programme by turning the TV set on could be thought of as an interaction in the realization environment, if the programme has been rendered for the current view, and the interaction is to start playing the programme. Alternatively, the programme can be thought of as existing in the construction environment, and the effect of starting to play the programme is that each frame is then transformed through the virtual, viewing, logical and realization environments to appear on the screen of the TV set in the current location with respect to the room and to be rendered through the current view and other transformations of the logical and realization environments. To take into account the effects of the TV programme on the lighting within the room requires that the whole scene be rendered for each frame in the programme. This suggests that the rate of redisplay is dictated by the frame rate of the TV programme. In more complex situations where a number of TV sets are introduced into the room with differing frame rates, the redisplay task can become more demanding.

The room, TV set, and TV programme are concerned with the arrangement of the presentation of the room in both space and time. The TV programme object manages the presentation of the programme on the TV set. This object needs to start and stop the programme in response to operator interactions through the on/off button (this is event–based synchronization) and to synchronize the video and soundtrack (both audio and subtitle) presentations as the programme is presented frame by frame.

To ensure correct synchronization between audio, video and subtitles, reference points are defined within the audio's coordinate system with corresponding synchronization elements, whose effect is to invoke operations that produce the required progression through the video and subtitle sequences.

The TV set object organizes the layout of the case, the on/off button and the TV programme on the set. This object also turns the TV programme on and off through events generated in response to operator interactions with the on/off button. This involves cooperation with the TV programme object. There is a constraint that the on and off events can only be generated when the TV set is located within the boundaries (spatial and temporal) of the room.

The timing diagram on Figure 5 illustrates the timing and synchronization constraints resulting from the example's object layout. The timelines of each of the constituent objects are mapped through the transformation hierar-

chy into the timeline of the room object. The diagram illustrates synchronization of the TV programme. There are synchronization events generated by the audio object to synchronize the audio and video presentations. Synchronization events are generated at a lower frequency to synchronize the subtitles with the audio and video presentations. On and off events correspond to the operators action in turning the TV set on and off. The viewer may enter and leave the room and corresponding events may be raised when these situations occur, for example, we might have an intelligent room in which the lights are turned off when the room is unoccupied!

Operator interaction can be defined with respect to a media object's content. In the example, interaction is allowed with the on/off button of the TV set. We assume that the manipulation process in the virtual environment will at some stage receive input tokens passed up through the lower level environments. These are generated by the operator manipulating some physical input device — the manipulation process in the realization environment will have performed some form of physical to logical input device mapping. The input tokens received by the virtual environment represent on and off events of the TV set's button. The virtual environment's manipulation then generates an on event or off event which is received by the TV set. The TV set invokes 'play' on the TV programme, which consequently invokes 'play' on each of its children.

6. Conclusion

Large classes of applications require standard tools to develop application–specific multimedia environments involving the merge, the transformation, the transmutation, and the integration of various media. This standard should focus on the presentation and programming aspects of multimedia. The emerging PREMO standard should provide such an environment, combining low–level multimedia resource management (offered by MSS) with high–level modelling concepts (offered by Part 3, i.e., CGRM, and the additional components developed on the basis of Part 3). Once available as an International Standard, PREMO should have a significant influence on the development of multimedia application in the field of, e.g., simulation systems, virtual reality environments, scientific and engineering visualization systems.

Acknowledgement

Obviously, PREMO is a teamwork project, involving a large number of experts from a number of industrial and academic institutions involved in ISO/IEC SC24 WG6. Instead of trying to list everybody and thereby incurring the danger of forgetting and perhaps offending somebody, we prefer to omit such a long list. We would just like to express our gratitude to all the members of the ISO/IEC SC24 WG6 rapporteur group and others who have contributed to this work

References

1. I. Herman, G.J. Reynolds, and J. Davy, "MADE: A multimedia application development environment", *Proc. of the IEEE International Conference on Multimedia Computing and Systems, Boston (ICMCS'94)*, L.A. Belady, S.M. Stevens, and R. Steinmetz, eds, Los Alamitos, 1994. IEEE CS Press. Also in: Reports of the Centre for Mathematics and Computer Sciences (CWI), CS–9360, October 1993.

2. International Organisation for Standardization, *Information processing systems — Computer graphics — Programmer's Hierarchical Interactive Graphics System (PHIGS) (ISO IS 9592)*, 1988.

3. International Organisation for Standardization, *Information processing systems — Computer graphics — Programmer's Hierarchical Interactive Graphics System (PHIGS) — Part 4, Plus Lumière und Surfaces (PHIGS PLUS) (ISO DIS 9592–4)*, 1991.

4. International Organisation for Standardization, *Information processing systems — Computer graphics — Metafile for the storage and transfer of picture description information (ISO IS 8632)*, 1987.

5. International Organisation for Standardization, *Information processing systems — Computer graphics — Presentation environment for multimedia objects. Part 1: Fundamentals of PREMO, (PREMO); ISO/IEC 14478-1*, April 1995.

6. International Organisation for Standardization, *Information processing systems — Computer graphics — Presentation environment for multimedia objects. Part 2: Foundation Component, (PREMO); ISO/IEC 14478-2*, April 1995.

7. International Organisation for Standardization, *Information processing systems — Computer graphics — Presentation environment for multimedia objects. Part 3: Modelling, Rendering, and Interaction Component, (PREMO); ISO/IEC 14478-3*, April 1995.

8. International Organisation for Standardization, *Information processing systems — Computer graphics — Presentation environment for multimedia objects. Part 4: Multimedia System Services, (PREMO); ISO/IEC 14478-4*, September 1994.

9. International Organisation for Standardization, *Information processing systems — Computer graphics — Graphical Kernel System (GKS) functional description (ISO IS 7942)*, 1985.

10. International Organization for Standardization, *Information processing systems — Computer graphics — Graphical Kernel System (GKS) functional description (ISO/IEC 7942-1:1994)*, 1994.

11. F. Arbab, I. Herman, and G.J. Reynolds, "An object model for multimedia programming", *Computer Graphics Forum (Eurographics'93 Conference Issue)*, 12(3):C101–C114, September 1993. Also in: Reports ofthe Centre for Mathematics and Computer Sciences (CWI), CS–9327, April 1993.

12. V. de May, C. Breiteneder, L. Dami, S. Gibbs, and D. Tsichritzis, "Visual composition and Multimedia", *Computer Graphics Forum (Eurographics'92 Conference Issue)*, 11(3):C9–C21, September 1992.

13. V. de May and S. Gibbs, "A multimedia component kit", *Proc. of ACM Multimedia'93*, P.V. Rangan, ed., Anaheim, CA, August 1993, ACM Press, pp 291–300.

14. "Common Object Services Specification (COSS)", Version 1.0, Technical Report, Object Management Group, March 1993.

15. J. Wernecke, *The Inventor Mentor.* Addison Wesley, Reading, Massachusetts, 1994.

16. A. Lie and N. Correia, "Cineloop synchronization in the MADE environment", *Proc. of the IS&T/SPIE Symposium on Electronic Imaging, Conference on Multimedia Computing & Networking, 1995*, San Jose, February 1995.

17. International Organization for Standardization, *Information processing systems — Computer graphics — Computer Graphics Reference Model (CGRM) (ISO/IEC IS 11072)*, 1 edition, 1992.

18. G.S. Carson, (ed.), "Introduction to the Computer Graphics Reference Model", *Computer Graphics*, 27(2):108–119, September 1993.

19. A. Diller, *Z: An introduction to formal methods*. John Wiley & Sons, Chichester — New York — Brisbane — Toronto — Singapore, 1990.

20. B. Potter, J. Sinclair, and D. Till, *An Introduction to Formal Specification and Z.* International Series in Computer Science. Prentice Hall, New York London Toronto Sydney Tokyo Singapore, 1991.

21. R. Duke, P. King, G. Rose, and G. Smith, “The Object-Z specification language: Version 1”, Technical Report 91-1, The University of Queensland, Queensland, Australia, April 1991.

22. G.J. Reynolds, D.A. Duce, and D.J. Duke, “Report of the ISO/IEC JTC1/SC24 Special Rapporteur Group on Formal Description Techniques”, ISO/IEC JTC1/SC24 N1152, June 1994.

23. International Organization for Standardization, *Information Processing Systems — Open Systems Interconnections — LOTOS (Formal Description Technique based on the temporal ordering of observational behaviour) (ISO/DIS 8807)*, March 1988.

24. D.A. Duce, D.J. Duke, P.J.W. ten Hagen, I. Herman, and G.J. Reynolds, “Formal methods in the development of PREMO”, *Computer Standards & Interfaces*, 16, 1995 (to appear). Also in: Reports of the Centre for Mathematics and Computer Sciences (CWI), CS–R9465, December 1994.

25. D.R. Begault, *3D Sound for Virtual Reality and Multimedia*. Academic Press, Boston, 1994.