Premo: An ISO Standard for a Presentation Environment for Multimedia Objects

To Cor Baayen, at the occasion of his retirement

I. Herman, P.J.W. ten Hagen, G. Reynolds
CWI
Department of Interactive Systems
Kruislaan 413, 1098 SJ Amsterdam, The Netherlands
{ivan,paulh,reynolds}@cwi.nl

PREMO is a major new ISO/IEC standard for graphics and multimedia, which addresses many of the concerns that have been expressed about existing graphics standards. In particular, it addresses the issues of configuration, extension, and interoperation of and between PREMO implementations. This paper gives an overview of PREMO and highlights its most significant features.

1 INTRODUCTION

The Graphical Kernel System GKS[1] was the first standard for computer graphics published by the International Organisation for Standardisation (ISO). It was followed by a series of complimentary standards, addressing different areas of computer graphics. Perhaps the best known of these are PHIGS[2], PHIGS PLUS[3], and CGM[4]. More recently, GKS[5] has been revised. These standardised functional specifications have had reasonable success either via direct implementations or through the influence they have had on the specification and development of other graphics packages (the most notable of this second category being the 3D extension of the X Window System, PEX[6, 7], which is largely based on PHIGS PLUS).

In spite of important differences in their functionality, these standards 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. However, 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 subcommittee responsible for the development and maintenance of graphics standards (ISO/IEC JTC1/SC24) recognised the need to develop a new line of graphics standards, along radically different lines from previous methods. To this end, a new project was started at an SC24 meeting at Chiemsee, Germany, in October 1992. Subsequent meetings (New Orleans, USA, January 1993; Steamboat Springs, USA, June 1993; Manchester, UK, November 1993; Amsterdam, The Netherlands, March 1994; Bordeaux, France, June 1994) resulted in a Draft for a new standard called PREMO (Presentation Environment for Multimedia Objects)[8]. 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 term "Presentation Environment" is of utmost importance in the specification of the scope of PREMO. 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 standardised, hence conceptually portable, development environment that helps to promote portable graphics and multimedia applications. PREMO concentrates on *presentation techniques*; this is what primarily differentiates it from other multimedia standardisation projects.

One of the main differences between PREMO and previous standards within SC24 is the inclusion of multimedia aspects; hence this activity is of importance for both the multimedia and graphics communities. The purpose of this paper is to present the motivation behind the development of PREMO, its major goals, and its relationship to other multimedia standards. An overview of the architecture of PREMO is given, although much of the detail is still subject to changes that result from the technical review process within ISO.

2 MOTIVATION

Three requirements have shaped the architecture of PREMO:

- the appearance of new media;
- the need for configurable and extensible graphics packages;
- the requirements of distributed environments.

2.1 Incorporation of Various Media

Traditional computer graphics systems and graphics applications have primarily been concerned with what might be called the presentation of *synthetic graphics*, *i.e.*, displaying pictorial information, typically on a screen or paper. The aims of any two presentations may be very different. Two characteristic examples are:

- produce photorealistic images (*e.g.*, in commercial film production, or high quality animation) using very complex models describing the surrounding reality;
- produce ergonomically sound and easy–to–grasp images of complex computed or measured data (*e.g.*, in scientific visualisation, or medical imaging).

These aims determine different fields of interest within computer science, which are all referred to under the heading of "computer graphics" and which are all to be addressed by PREMO.

Developments over recent years have, however, resulted in new applications where synthetic graphics *in isolation* cannot cope with the requirements. Technology has made it possible to create systems which use, within the same application, different presentation techniques that are not necessarily related to synthetic graphics, *e.g.*, video, still images, and sound. Examples of applications where video output, sound, etc., *and* synthetic graphics (*e.g.*, animation) coexist are numerous and well–known. It is therefore a natural consequence to have development environments that are enriched with techniques supporting the display of different media in a consistent way, and which allow for the various media–specific presentation techniques to coexist within the same system.

"Coexistence" is not enough, though; *integration* is also necessary. For example, an audio display is not necessarily independent from the (synthetically generated) image being displayed: the viewer's position in the model, or indeed the model itself when displayed, may influence the attributes of audio presentation. This influence may be very simple (*e.g.*, the volume may depend on the distance from the viewer), but it may also require very complicated sound processing techniques (*e.g.*, to take the acoustic properties of the room model into account for sound reflection and absorption). In other words, it should be possible to describe media objects *integrated* with geometry and with one another, and also to describe and control their mutual influence. The complete integration of various media and their presentation techniques within the same consistent framework is one of the major goals (and challenges) of PREMO, and one of the features which will make it very different from earlier SC24 standards, and indeed, other multimedia standards that are either already available or under development (such as HyTime[9], HyperODA[10], and MHEG[11]).

The introduction of new media brings new problems for PREMO that, hitherto, have been unknown in earlier SC24 standards. One of the most intricate issues of some importance is that of *synchronisation*, *e.g.*, synchronisation of video and sound presentation. This problem is well–known in the multimedia community; its integration with the more general demands of a presentation system will obviously be a challenge.

2.2 Configurable and Extensible Graphics Packages

As mentioned in §1, most traditional ISO graphics packages, as well as the majority of graphics systems available on the market-place, are defined as

monolithic libraries containing large sets of functions with precisely defined semantics. These libraries are frequently referred to as *kernels*. The choice of functionality for a specific kernel reflects the particular application areas which the kernel tries to address.

Modifying and extending the existing functionality of a kernel requires the definition of additional sets of functions. These functions may either add to or modify existing behaviour. However, modification of the standard interface is not allowed, which often means that these new definitions form completely separate packages on top of the standard with their own sets of well–defined functions.

This rigidity of current ISO graphics standards is in a sharp contrast with the extraordinary diversity of the algorithms used in computer graphics, in visualisation, and in other related application areas. Radically new visualisation techniques are developed, old and apparently well–established algorithms are constantly re–visited. This diversity and fervent activity is very well reflected in the proceedings of the major computer graphics and visualisation conferences worldwide (such as, for example, the ACM SIGGRAPH, Eurographics, and Nicograph annual conferences and workshops, IEEE's Visualisation conferences, etc.).

As a consequence, major rendering techniques, which are almost commonplace in advanced graphics applications, cannot be integrated into SC24 standards; the most startling examples being ray–tracing and radiosity. Although these graphics standards include a rudimentary mechanism to add new graphics primitives, for example in the form of the GDP, (Generalised Drawing Primitives), this mechanism does not give the full power needed by a number of applications to add new display algorithms and/or to modify some aspects of the ones included in the package in use[1].

Note that the inclusion of different media into a new standard makes this type of problem more acute. The techniques to achieve integration of media are extremely disparate, and they use the results of various fields of computing technology, like, for example, high quality synthetic graphics, image processing, speech synthesis, etc. Some of the techniques are also application dependent. It is almost impossible to define a closed programming environment which would satisfactorily encompass all these needs; even if a specification could be finished, complete implementations would be so complex that the entire product would lag behind current technology.

The usual approach to solve such problems is to use object–oriented techniques. This is also the approach that has been adopted by PREMO. Object–oriented techniques have already been used for graphics and for multimedia, and they have proven their values in using inheritance as a tool for extensibility and user configurability (see, *e.g.*, [12, 13, 14, 15, 16]). Using inheritance, additional information may be integrated into an existing object of a graphics system, allowing extensive reuse of inherited methods. Referring to the

[1]Escapes also offer some possibilities for modifying algorithms in a restricted way, but such extensions lead away from portability.

example above, in a carefully designed object–oriented system it would be possible to redefine the reflection equations of a "shader object" only, and thereby make full use of the power of the surrounding system with the shading method adapted for a particular use.

2.3 Distribution

It is no longer necessary to argue in favour of distributed environments; their widespread availability has made their use very natural in both academia and industry. Some graphics and multimedia applications and tools are notoriously computationally intensive, and as such are prime candidates to exploit the advantages offered by a distributed environment.

There have been numerous projects in the past which have tried to use, *e.g.*, GKS or PHIGS in a distributed setting; it was never easy. Indeed, the SC24 graphics standards were not particularly well prepared for distribution (see, for example, [17, 18, 19]). In contrast, and using the terminology which has become widespread in the past years, particular PREMO implementations may offer multimedia or graphics "services" on a network; hence, the PREMO specification should allow for the straightforward implementation of such services.

Object–oriented technology also provides a framework to describe distribution in a consistent manner. Objects can be considered as closed entities which provide "services" via their methods; from the point of view of the object specification it is immaterial how an object method is realised: within the same program, or via calls across a network.

Defining complex object–oriented systems to be used in a distributed environment leads to software engineering issues, whose complete solution would go far beyond the charter (and the experiences) of the PREMO working group. Instead, the PREMO specification will make use of techniques developed elsewhere, both within and outside ISO. Currently, another ISO working group (ISO/IEC JTC1/SC21 WG7) is working on what is called the "Open Distributed Processing Initiative" (ODP); PREMO intends to rely on the experiences of this working group, and include their results into the PREMO document proper. The goal is to develop a specification which would be compliant with ODP. A liaison agreement has also been set up with the Object Management Group[2] (OMG), whose CORBA specification[20] has already influenced the current design of PREMO.

3 General Architecture

Underlying all of PREMO is a concise conceptual framework, comprising a description technique (not detailed here), an abstract object model used for the definition of data types and the operations upon them, and the notion of components which contain and organise the PREMO functionality needed to address specific problem areas.

[2]The Object Management Group is primarily an industrial consortium established to define a unifying model amongst/from a number of emerging object technologies.

3.1 The Conceptual Framework

The conceptual framework addresses three fundamental areas: an object model, the activity of objects, and events and event handling.

3.1.1 Object Model

At the earliest stages of the PREMO project specification it became clear that a concise framework, *i.e.,* a precise *object model*, would be needed to ensure the smooth cooperation among objects within PREMO and also to provide a consistent approach to some of the technical issues raised by multimedia programming in general. Such an object model was adopted at an early stage of the PREMO project. This object model is traditional, being based on subtyping and inheritance. The PREMO object model supports both multiple supertypes and multiple inheritance.

As said earlier (c.f. §2.2), subtyping and inheritance provide the basic mechanism in PREMO for extensibility and configurability.

In PREMO, a strong emphasis is placed on the ability of objects to be active. This feature of PREMO stems from the need for synchronisation in multimedia environments (§2.1). 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 synchronisation in general terms as well (see also [14, 15, 16] 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 for PREMO, in contrast with the OMG object model. 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 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, *e.g.*, locator position values. A separate object modelling (or directly interfacing) a locator can send thousands of motion notification messages to a receiver object, and this latter can just "sample" these messages using the sampled message facility.

Using active objects, synchronisation appears to be no more and no less than synchronisation of concurrent processes, *i.e.*, concurrent active objects in PREMO. This does *not* mean that synchronisation becomes easy. What it *does* mean is that the terminology, the results, the machinery, etc, of the theory and the practice of concurrent programming can be reused in PREMO. There are other issues of synchronisation that can be considered *quality of service* issues, which go beyond this basic synchronisation model. Nevertheless, the model provides a clean and straightforward framework on which other such facilities can be built.

3.1.2 Events, Event Model

The PREMO framework includes the notion of non–objects, primarily for efficiency reasons. Non–objects have no requests defined on them, they cannot take part in subtyping and inheritance hierarchies.

Events form a special category of PREMO non–object types, and are the basic building block for the PREMO event model. Events and their propagation (described by the event model) play a fundamental role in the synchronisation mechanism.

The event model is based on three concepts: events, event registration, and event handling. An *event* can model any action that occurs at a definite time. Events are created by *event sources*, and are consumed by *event clients*, both of which are objects. A basic characteristic of an event is its distinct type, which is one of the characteristics that a client uses to identify the events in which it is interested.

Whereas in object communication, the caller specifies the recipient of each operation request, in event communication, events are not addressed to specific recipients. Instead, it is the recipient that determines which events it wishes to receive. An object can register interest in receiving specific events produced by the various objects. As part of the registration process, a client can specify one of its (asynchronous or sampled) methods to receive events forwarded by an `Event Handler` object (defined by the so–called Fundamental Component, see §4.1). Prospective event recipients specify which events they are interested in by registering constraint lists with an `Event Handler`. Each constraint list defines the event names and parameter values which the event recipient wishes to receive. In the most common case, the constraint list specifies the name of an event in which the object is interested. Issuing an event by the event source means sending a message to an `Event Handler` object which dispatches the event to the interested event clients.

3.2 Components

The object model, the event model, the concept of non–objects, etc., described in §3.1, give a conceptual framework for all the basic notions in PREMO. *Components* allow for a structuring of the PREMO standard in terms of the services provided.

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 a 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* as in OMG (see §2.3), *i.e.*, services usable in a distributed environment, or it may be used as a set of objects directly linked to an application.

Components may be organised in component inheritance hierarchies. For example, in Figure 1, both components `B` and `C` inherit from component `A`. This means that object types in `B` and `C` are subtypes of types defined in `A` (see §3.1.1). All PREMO objects are subtypes of a common PREMO supertype, so this rule enables new types of objects to be defined. As far as subtyping and/or inheritance are concerned, objects within components `B` and `C` are all distinct types: no type in `B` may be a subtype of a type in `C` and vice versa.

The rule on component inheritance does not imply that objects in different components have to have a subtyping relationship in order to be able to communicate with one another. Again referring to Figure 1, `B` can of course make use of the *services* offered by component `C`. Components may also specify how they exploit functionality from other components, with the option of hiding this from the client. Hence components may become clients of other components' services.

Underlying all PREMO components is a *Foundation Component* providing functionality which is necessary for all PREMO components. It is mandatory that all other PREMO components inherit from this Foundation Component (described in more details in §4.1).

The rules for components are part of the standard. These rules form the basis, in conjunction with the object model, for the properties of configuration, customisation, extension, and interoperation.

4 COMPONENT STRUCTURE

With the above description of the conceptual framework and the component model, we now describe the structure of the PREMO standard in more detail.

The initial PREMO standard will:

- define the exact conceptual framework for multimedia presentation, along the lines described in §3.1, *i.e.*, the object model, the event model, etc.;
- define rules for components, their interrelationships, inheritance, conformance rules, etc.;

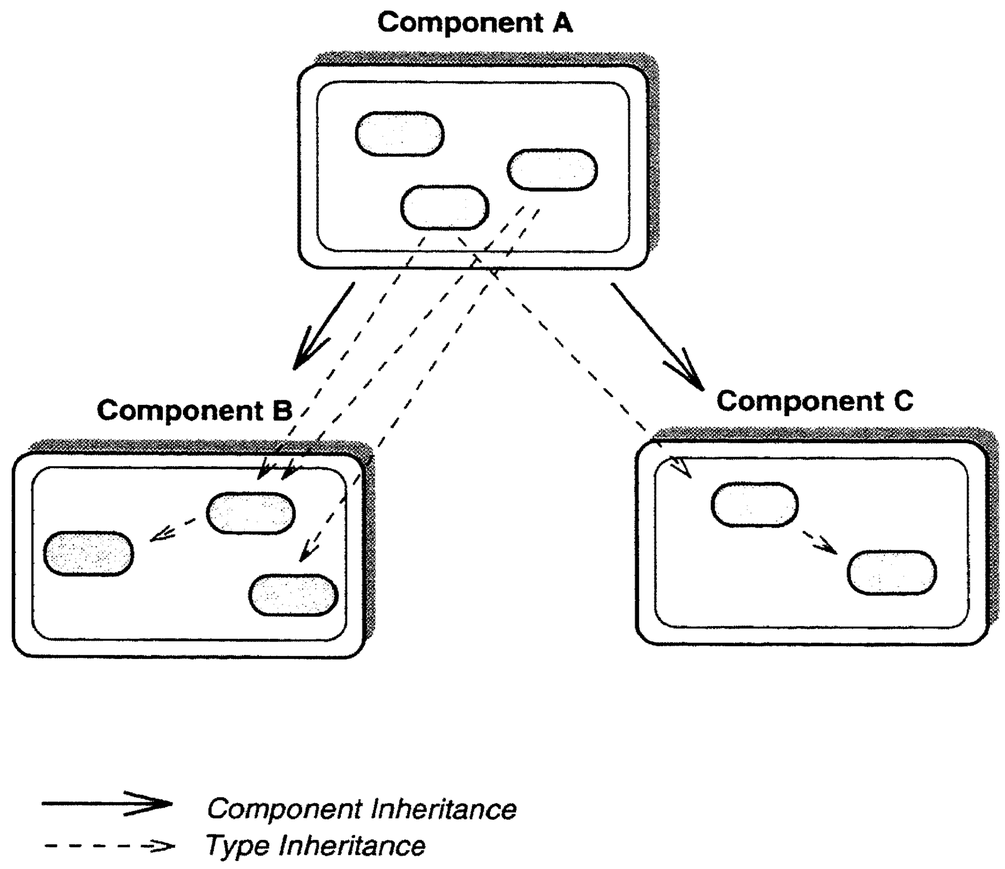

FIGURE 1. Component inheritance.

- include the specification of the Foundation Component;
- include the specification of some other components, namely:
  - a component for Multimedia System Services (see §4.2);
  - a Modelling, Presentation, and Interaction Component, which will provide for the basis of components inherently related to modelling, geometry, traditional computer graphics, etc.

PREMO should, however, be thought of as an evolving standard; new components will be added in the future. On the basis of the Modelling, Presentation, and Interaction Component, components may also be added to ensure applications using current SC24 standards will continue to work, and be upwards compatible. Two types of components are planned: expression of existing SC24 standards as PREMO components, *e.g.*, PHIGS or GKS, or new components, *e.g.*, a pure audio component, or a component for virtual reality. Although the exact component hierarchy is not yet finalised (June 1994), Figure 2 gives a view of the expected hierarchy of standardised components.

In the following sections, highlights of some of the components referred to above are given. The reader should remember, however, that the specification of these components is still an ongoing activity.

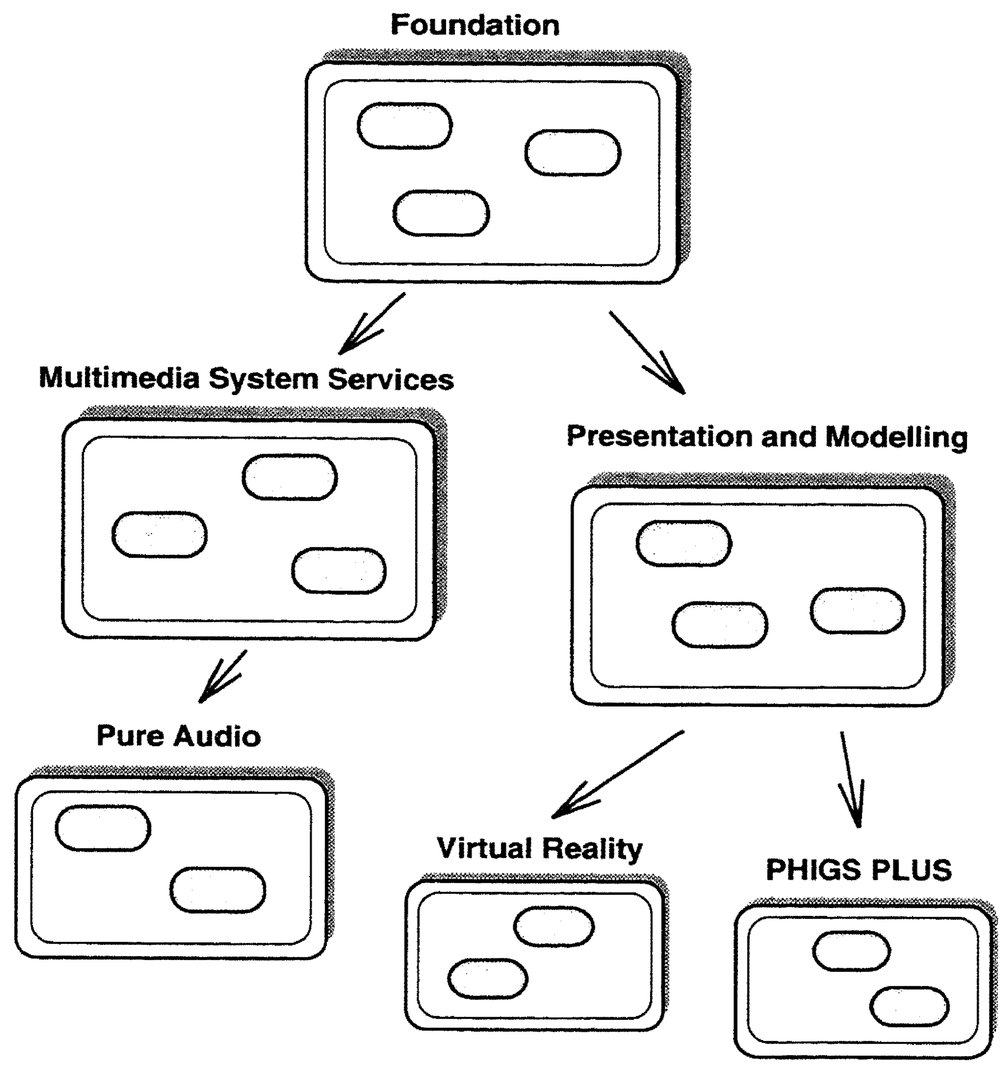

FIGURE 2. Component hierarchy.

4.1 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. The list of foundation objects includes the following object types:

- The PREMO `Life-cycle Manager` object provides object life cycle services for PREMO objects. This includes the creation of new objects, destruction of object and object references, keeping track of object references. The separate management of object life–cycles and associated object references is essential if a component intends to offer services in a distributed environment.
  In fact, PREMO defines two such life cycle manager objects, whose functionalities are identical, but they manage remote, service objects and local object respectively. This distinction is necessary to control objects which

offer services over, e.g, a distributed environment and, alternatively, to have objects which are to be used in a local setting only.

- `Data` objects. The semantics associated with a data object 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.

- `Producer` objects provide an encapsulation for defining the processing of `Data` 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 type `Producer` may place restrictions on the number of sources and destinations of `Data` objects if necessary. Specific types of `Producer` object are characterised by the behaviour made visible through their associated sets of operations.

- A `Porter` object is the PREMO foundation object which interconnects to systems and environments defined outside of PREMO, e.g., files, physical devices.

- 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.

- `Event Handler` objects provide methods to register interest in certain events, for dispatching events to the interested objects, manage constraint lists for events, etc. These objects also play a fundamental role in synchronisation mechanisms.

As an example of how these notions can be used, let us see how basic, event–based, synchronisation can be expressed with these objects. Synchronisation is handled by using synchronisation events that are sent by synchronisation sources to event handlers. An `Event Handler` then forwards the event to objects that have registered their interest in these events. The interested objects could be either objects that are the immediate target in the synchronisation, or controller objects for more elaborate synchronisation. Figure 3 illustrates a more complex case: two `Event Handlers` take care of two independent clock events, but, for one of them, the same event may also be "simulated" by another object. A separate controller receives these events and, based on its own internal state, may then dispatch a synchronisation call to two other PREMO objects.

The combination of `Event Handlers` and `Controllers` can also be used for schemes where actions are scheduled to take place at a certain time. In this

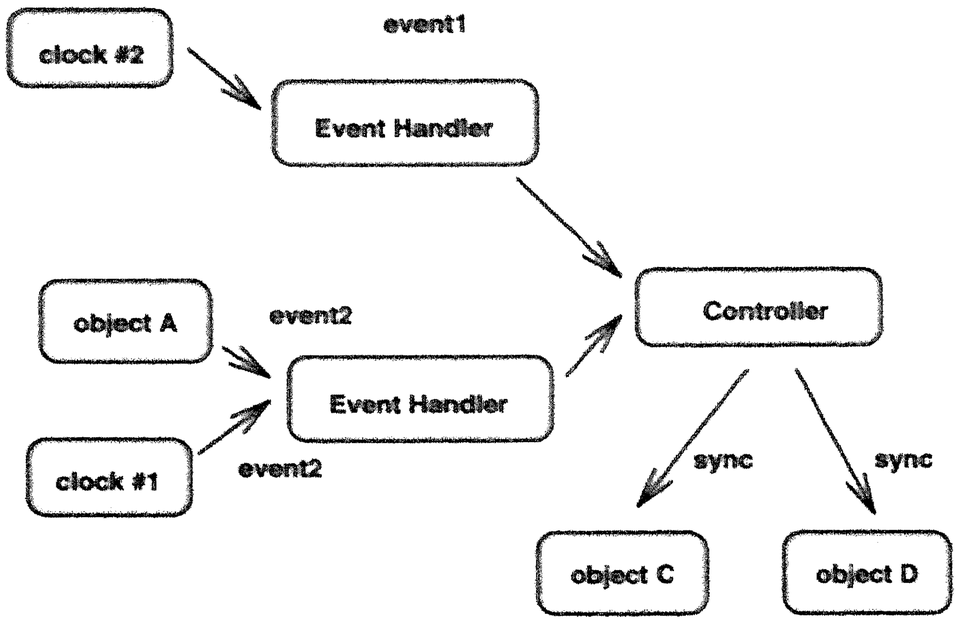

FIGURE 3. Synchronisation.

case, a clock object (to be provided by a higher level component) can be used to trigger the action at the right time. This allows for the more general notion of temporal synchronisation.

4.2 Multimedia System Services

The primary goal of the Multimedia System Services (MSS), defined as a recommended practice by the IMA (Interactive Multimedia Association), is to provide an infrastructure for building multimedia computing platforms that support interactive multimedia applications dealing with *synchronised, time-based*, media in a heterogeneous distributed environment. The emphasis is very much on distributed services for "low level" media processing; MSS does *not* include any concepts for geometry, modelling, etc. Instead, it is concerned with problems like the definition of abstract media devices, resource control, connections among virtual devices (in the form of so-called streams), etc.

Active cooperation between the ISO PREMO group and IMA and resulted in the decision encapsulate MSS within PREMO. Figure 2 shows how MSS will be integrated into PREMO: it will form a separate component, relying on the objects defined in the Foundation Component. The design of these objects already reflects the requirements of MSS. A first implementation of MSS will be available (independently of PREMO) in the course of 1995, and the first draft for an integration with PREMO will be available in 1996.

4.3 Presentation, Modelling, and Interaction Component

The Presentation, Modelling, and Interaction Component (PMI) of PREMO 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 the Presentation, Modelling, and Interaction Component, might refine the renderer objects of the PMI component to objects most appropriate in the virtual reality domain. This component introduces abstractions for such things as modellers, modelling objects and their properties, scenes, renderers, etc. Objects with geometry may be placed into scenes, and may subsequently be transformed and visualised. This notion is a general one and applies equally well to objects that do not have a clear graphical representation. For example, an audio object with spatial properties can be located within a scene and appropriate rendering algorithms can take this into account to achieve a stereo audio effect. The abstractions defined in this component will also allow for the inclusion of objects with time properties.

The Presentation, Modelling, and Interaction Component of PREMO heavily relies on an existing reference model, called the Computer Graphics Reference Model (CGRM)[21], developed within the same ISO group (ISO/IEC JTC1/SC24) some years ago. In fact, the PMI could be viewed as the adaptation of CGRM (which is an abstract framework) to the object oriented environment defined by PREMO

Based on the Presentation, Modelling, and Interaction Component, more "concrete" components will be developed. Activities have already started on the development of a Virtual Reality component, and other possibilities (*e.g.*,, pure audio component, solid modelling component) are currently explored.

5 A Formal Approach to Developing the PREMO Standard

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 RG plans to address this problem by employing formal methods at an early stage and to continue this activity throughout PREMO's development. This task started after the July 1993 PREMO meeting and some early results are documented in [22, 23]. 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 used is based on Z[24] and Object-Z[25].

6 Timetable

The current timetable for the work progress in PREMO is as follows:

Draft International Standard:	June 1996
International Standard final text:	June 1997

7 Experimental Implementations

In the near future, work will also begin on an experimental implementation of the PREMO standard. The major emphasis of this work will be to provide a proof of concepts for the main paradigms and the models advocated by the PREMO document. The implementation of the object model will require a major effort; indeed, the requirements of this model go far beyond what is offered "by default" by languages like C++[26]. Fortunately, tools already exist which will make this activity easier. The environment developed within the ESPRIT MADE project[27], primarily its object model implementation[16], will be used as the basic tools to develop a first, experimental implementation of PREMO. [3]

Acknowledgements

Obviously, PREMO is a teamwork project, involving a large number of experts from a number of industrial and academic institutions involved in ISO/IEC JTC1/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 JTC1/SC24/WG6 rapporteur group.

References

1. International Organisation for Standardisation, Geneva, *Information processing systems — Computer graphics — Graphical Kernel System (GKS) functional description (ISO IS 7942)*, 1985.
2. International Organisation for Standardisation, Geneva, *Information processing systems — Computer graphics — Programmer's Hierarchical Interactive Graphics System (PHIGS) (ISO IS 9592)*, 1988.
3. International Organisation for Standardisation, *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 Standardisation, Geneva, *Information processing systems — Computer graphics — Metafile for the storage and transfer of picture description information (ISO IS 8632)*, 1987.
5. International Organization for Standardisation, Geneva, *Information processing systems — Computer graphics — Graphical Kernel System (GKS) functional description (ISO/IEC 7942-1:1994)*, 1994.
6. W. Clifford, J. McConnell, and J. Saltz, "The development of PEX," in *Eurographics'88 Conference Proceedings* (D. Duce and P. Jancène, eds.), (Amsterdam), North–Holland, 1988.

[3] Note that elements of the MADE object model, *e.g.*,, the notion of sampled messages, have already significantly influenced the development of the PREMO object model.

7. R. Rost, J. Friedberg, and P. Nishimoto, "PEX: A network–transparent 3D graphics system," *IEEE Computer Graphics & Applications*, vol. 9, pp. 14–25, 1989.
8. International Organisation for Standardisation, *Presentation Environment for Multimedia Objects (PREMO); ISO/IEC 14478*, June 1994.
9. Internation Organisation for Standardisation, *Information Technology — Hypermedia/Time-based Structuring Language (HyTime), ISO/IEC 10744:1992(E)*, 1992.
10. International Organisation for Standardisation, Geneva, *Information technology — Open Document Architecture (ODA) and Interchange Format — Temporal relationships and non-linear structures (ISO/IEC DIS 8613-14:1993)*, 1993.
11. International Organisation for Standardisation, *Information Technology — Coded Representation of Multimedia and Hypermedia Information Objects (MHEG)*, ISO/IEC CD 13522 ed., June 1993.
12. P. Wißkirchen and K. Kansy, "The new graphics standard — object oriented!," in *Advances in Object–Oriented Graphics I* (E. Blake and P. Wißkirchen, eds.), EurographicSeminar Series, Berlin – Heidelberg – New York – Tokyo: Springer-Verlag, 1991.
13. M. Kaplan, "The design of the Doré system," in *Advances in Object–Oriented Graphics I* (E. Blake and P. Wißkirchen, eds.), EurographicSeminar Series, Berlin – Heidelberg – New York – Tokyo: Springer-Verlag, 1991.
14. V. de May, C. Breiteneder, L. Dami, S. Gibbs, and D. Tsichritzis, "Visual composition and Multimedia," *Computer Graphics Forum (Eurographics'92)*, vol. 11, no. 3, pp. C9–C21, 1992.
15. V. de May and S. Gibbs, "A multimedia component kit," in *Proceedings of ACM Multimedia'93* (P. Rangan, ed.), (Anaheim, CA), pp. 291–300, ACM Press, August 1993.
16. F. Arbab, I. Herman, and G. Reynolds, "An object model for multimedia programming," *Computer Graphics Forum (Eurographics'93 Conference Issue)*, vol. 12, pp. C101–C114, September 1993.
17. I. Herman, T. Tolnay-Knefély, and A. Vincze, "XGKS — a multitask implementation of GKS," *Computers and Graphics*, vol. 8, 1984.
18. G. Reynolds, "A token based graphics system," *Computer Graphics Forum*, vol. 5, pp. 139–145, June 1986.
19. D. Arnold and M. Hinds, "On implementing parallel GKS," *Computer Graphics Forum*, vol. 8, 1989.
20. Object Management Group, *The Common Object Request Broker: Architecture and Specification; OMG Document Number 91.12.1, Revision 1.1*, 1992.
21. Internation Organisation for Standardisation, *Introduction to the Computer Graphics Reference Model, ISO/IEC JTC 1/SC24 N849*, 1992.
22. International Organization for Standardisation, Geneva, *Report of the ISO/IEC JTC1/SC24 Special Rapporteur Group on Formal Description Techniques*, 1994.

23. D. Duce, D. Duke, P. ten Hagen, and G. Reynolds, "PREMO – an initial approach to a formal definition," *Computer Graphics Forum (Eurographics'94 Conference Issue)*, vol. 13, pp. C393–C406, September 1994.
24. B. Potter, J. Sinclair, and D. Till, *An Introduction to Formal Specification and Z.* International Series in Computer Science, New York London Toronto Sydney Tokyo Singapore: Prentice Hall, 1991.
25. R. Duke, P. King, G. Rose, and G. Smith, "The Object-Z specification language: Version 1," Tech. Rep. 91-1, The University of Queensland, Queensland, Australia, April 1991.
26. B. Stroustrup, *The C++ Programming Language.* Reading, Massachusetts: Addison-Wesley, second ed., 1991.
27. I. Herman, G. Reynolds, and J. Davy, "MADE: A multimedia application development environment," in *Proc. of the IEEE International Conference on Multimedia Computing and Systems, Boston (ICMCS'94)* (L. Belady, S. Stevens, and R. Steinmetz, eds.), (Los Alamitos), IEEE CS Press, 1994.