Distributed Objects: an approach to sharing multimedia information

Paulo Pinto and Luis Bernardo
Inesc/IST, R. Alves Redol, 9 P-1000 Lisboa Portugal
{pfp,lflb}@inesc.pt

Abstract: *This paper addresses the issue of multimedia cooperative work. An ODP based view of the system is discussed taking into account some of the viewpoints of the standard. Special emphasis is given to the modeling of multimedia stream interfaces and the concept of binding object which is a key concept. The paper describes a multimedia distributed platform, DIMPLE, supporting interactive and distributed multimedia applications. The example application implemented allows for a shared synchronized access to multimedia information systems and includes an off-the-shelf videoconferencing system which is a non-ODP product. It is shown that the concept of binding object makes the integration very easy.*

1. Introduction

Handling multimedia data on networked and distributed systems begins to be a known issue with a clear identification of the problems still open. Most of the work is centered on videoconferencing systems ([1], [15], or commercial products such as ShowMe and Communique) and high-level synchronization and presentation aspects on information retrieval ([8], [10], [16], [12]). Major problems still open are: low-level synchronization of real-time data when using asynchronous networks [2],[14]; definition of relevant quality of service properties for the multimedia case; and appropriate application concepts to model multimedia.

Another area of interest to this paper is cooperative work (CSCW). Cooperative work is achieved either by using shared aware applications [15], or shared unaware ones making use of certain transparencies given by utilities such as SharedX. Systems are, such as Berkom or ShowMe, which use a videoconferencing system and a shared working space based on SharedX. The limitation of the SharedX approach is that it works at rendering level, making it unsuitable to be used by multimedia streams.

This paper describes a framework using the ODP concepts [9] for sharing multimedia information, based on distributed objects. If the concept of a user in the system is not well chosen, it will create unnecessary complexity both at communication and control levels each time a user wants to join or leave the system.

A typical application might be a shared multimedia retrieval system in which users can see the same information at the same time and browse as if only one user was in the system. Furthermore, users can join the session (and leave) when they please getting the current state as everybody else in the system at that time. A useful purpose might be applications in which tutors supervise the learning process. For instance, the replacement of a broken piece on an industrial machine. The maintenance person would learn from a multimedia document. If (s)he has some doubts (s)he could invite the support responsible person from the manufacturer to the session. The support person would follow the session; explain further details using the videoconferencing; or influence some browsing decisions. The support person would join exactly at the place where the problem was and could leave anytime after that (joining again, etc.).

The system was implemented using the version 4.1 of ANSAware [3] with some system add-ons to implement the engineering objects required. The videoconferencing tool used was the Berkom system that provided an example of connection with the non-ODP world (Berkom uses the OSI stack of protocols for control and UDP protocols for real-time data transport).

This paper starts with a brief description of the general structure of the distributed multimedia platform, called DIMPLE, used to construct applications. Only the necessary information to understand the rest of the paper is given and further details can be seen in [12] [13]. Section 3 describes how the computational concepts of ODP were used to fit the platforms´ requirements. The following section describes the engineering objects necessary to implement the computational ones, and the choice of protocols is the subject of section 5. Section 6 shows how a non-ODP system was nicely integrated as long as only communication is concerned. Section 7 describes an application for the scenario defined above, and the final section draws some conclusions and points some issues not yet fully defined which were left for further study.

2. General Description of the DIMPLE Platform

Multimedia applications can be created on the DIMPLE platform by defining control interactions between objects. Objects are distributed entities that manage entirely one medium, or several media in a bunch, and offer a unique point for control (They are called MMobjects). MMobjects rely on *component* objects to provide the functions of sources, sinks and filters (see figure 1). Control interactions are made of events and actions between objects, and their description is suitable to be written using a process algebras´ based language. Actions are simply submissions of invocations and all the returning values came as events. Texts, movies, pictures are modeled as processes that start and end, and can be composed with each other using the language operators. All important control information in objects is seen at platform level by events and actions. MMobjects show interest on events implicitly from the behaviour expressions in the language. There are two major classes of events: those related to the multimedia data called *deterministic events* (object started, ended, a video frame was played); and those related to any other feature in the system called *non-deterministic events* (user intervention, change of QoS of the object, change of volume or speed, etc.).

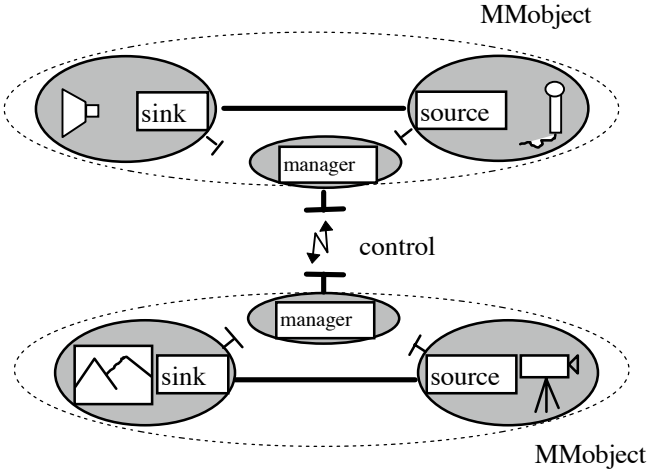

Fig 1 - MMobject interaction

DIMPLE was conceived to deal with multimedia and entities differ very much depending on their nature (e.g. audio has volume, video has geometry and colour, etc.). One approach to handle all aspects is the usage of operation overloading. However, this can be unnatural to programmers and application writers so groups of actions were joined together with their respective events and state data to form *statuses*. An MMobject is thus a collection of status relevant to the type of the multimedia data it controls. In terms of type systems (or type names

with compatibility relations) there are several: events; annotations (which are a special kind of events); MMobjects; components; statuses; actions; and multimedia data connection points.

The inclusion of new types of data, or the upgrade of existing ones while the system is running was seen as an important issue due to the field of application. It can also happen that new applications (the interaction description) would use old objects or the way around. This was achieved by an incremental type system [13] where type information is checked at run-time. Type safety exists because when objects support more operations than the application knows of these operations will not be called. On the other hand, new applications could try to submit an invocation on new operations not yet supported by old objects. This attempt is trapped by the platform because operation type checks are performed at run-time using type information from a Type Repository. It will fail but applications have to recover and simply do not interact in that way (action invocation can be triggered by the behaviour expressions of the language and not only from the compiled code). This latter feature is sustainable when new operations refer to non-basic synchronization issues (such as increasing the volume, or the speed of presentation). If operations fail the application will have a different behaviour because the objects used belong to older versions. When synchronization related actions have to be introduced, then a major version of the platform has to be created.

All the basic synchronization actions are present in all MMobjects and constitute the status *Life* and *Context*. Status *Life* has actions to *Prepare* and *Dismiss* objects; status *Context* has the actions *Play*, *Stop*, *Destroy*, *Inform* (of references to other statuses of the objects), *RegisterStatusInterest* and *UnregisterStatusInterest* on events of a certain status, and *ReceiveEvent* as a general dispatcher for events. Component objects also have a *Life* status (*LifeComp*) and a *Context* status. All other statuses are optional on MMobjects. The type of MMobjects reflects the components gathered to make it and the group of statuses it provides. The actions of the statuses can be implemented directly by the MMobject manager or delegated on the components of the object. For the clients this is transparent because all they get is a reference for the status operation interface.

3. Computational Description

One of the objectives of this paper was to test how suitable were the ODP concepts of the computational viewpoint to model multimedia applications. Therefore, the focus of this section will be mainly on this point. MMobjects on the platform consist of basic computational objects (BCO) and binding objects (BO). Basic computational objects model the interactions between MMobjects (and between the MMobject manager and the components) for the aspects of event handling and action invocation. Interrogations were used for both interactions although the objects simply collect terminations (a reliable mechanism for announcements would be enough). So, at the highest level of refinement an MMobject is a BCO. One level down, the MMobject manager and the components are themselves BCOs. Among other interactions they divide, cooperatively or not, the statuses offered by the MMobject.

The concept of binding object was used to model the multimedia connections between components. The binding object has an operation interface for control -- the *port* interface. Actually *port* is just as any other *status*, so there is the same computational concept for the language and execution -- e.g. objects get to know the reference to it using the parameter passing on operations. The control interface is important because it supports the explicit binding and the dynamic association (and separation) of connections over time. The BO also has stream interfaces for data. Figure 2 shows the components of the binding object with some insight already into the engineering concepts explained below. The external control is the *port*

interface with a specialization for internal control. The role interface is an engineering concept to model the local interface between the object and the channel.

A binding object is a distributed entity which controls communication between BCOs and has other control features related with the specific type of communication. These two issues will be described now.

With respect to the first issue, communication, explicit binding was used. BCOs get to know the interfaces' references of the objects they want connected. In other words, the MMobject manager gets the *port* control interfaces of its components and instructs the binding object manager. Thus, this stream binding is an operation performed by a third-party object. Polymorphism was used to define different types of stream interfaces. They all descend from *port* which has the *Connect* and *Disconnect* operations. Subtypes have low-level *connect* operations due to a reason explained below. The type space reflects three main issues: the communication protocol (TCP, UDP, XTP, etc.); the data encoding format; and the specific transfer protocol used (to maintain real-time, if the sink is pushing or pulling data, etc.). The compliance to just operation signature was seen as a weak verification here.

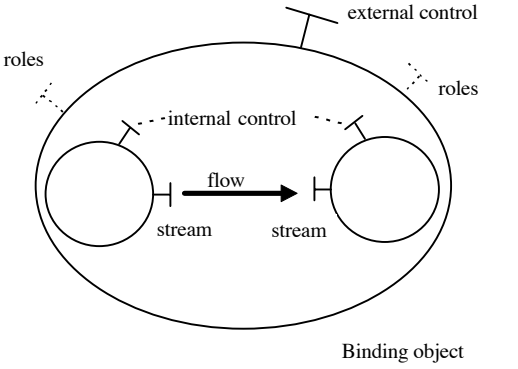

Fig. 2 - Binding object

Once the basic computational object gets the references for the connections it calls the *Connect* interface of the binding object. This call triggers, at engineering level, the low-level *connect* between the different objects that compose the BO to establish a stream interface. Unlike operation interfaces, stream interfaces use the connection-oriented paradigm, so explicit communication has to be performed to create resources. Operation interface bindings are simpler because servers are always ready to engage on invocation deliveries and do not need an explicit connection. Implicit binding could also have been used, and the first data would start the connection. This was not considered appropriate because it would put an additional delay on the first data, and resources could not be available at the time the data begins to be transferred.

A more detailed description of the binding sequence is useful at this point. The BCO MMobject manager invokes the *Prepare* action on each component. It gets to know the components by its own *Prepare* operation. The parameters for its *Prepare* are defined in the language. The language has two different parts (see figure 3): an ADT part defining the MMobjects, their properties (geometry, data files, etc.), their components, and their topologies; and a behaviour expression part which is the dynamic behaviour of the application (deterministic and not).

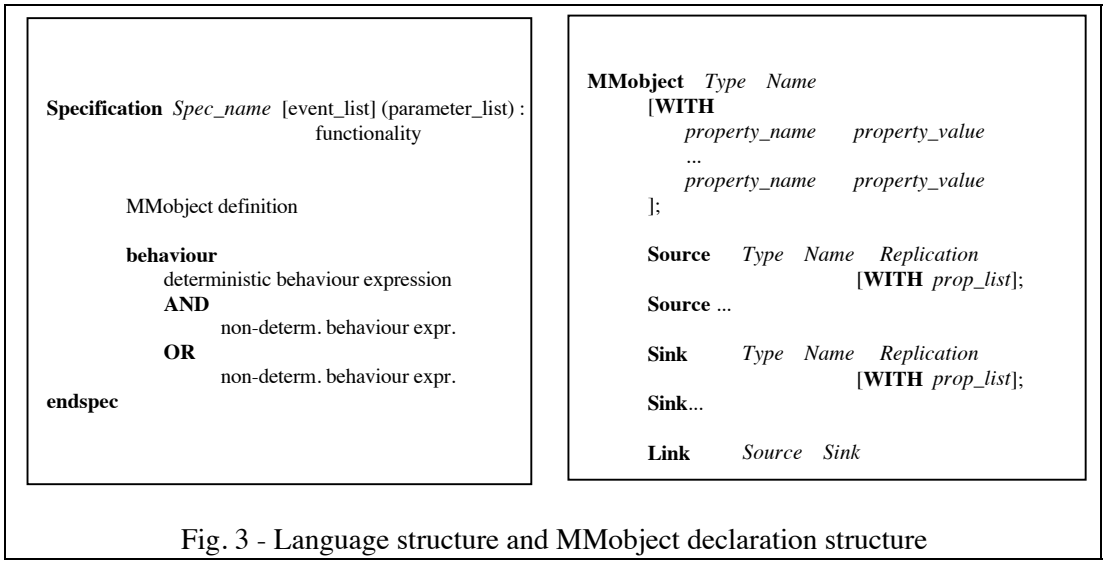

Fig. 3 - Language structure and MMobject declaration structure

Shared applications can create new connections dynamically to existing MMobjects. Topologies can fall into three different sets to model applications so different as videoconferencing, information retrieval, or mixing information: (a) new connections involve the creation of both sources and sinks; (b) just another sink connected to the same source; or (c) just another source connected to the same sink. This information is expressed on the *replication* parameter of sources (and sinks).

- **Source** *Type Name* means that a new connection uses the same source (e.g. a video source on a video on-demand application).
- **Source** *Type Name* [ ] means that a new connection will produce another source (e.g. a camera on a videoconferencing system)
- **Sink** *Type Name* [ ] [ ] means that a new connection will produce a new sink for all the connections established in the system (e.g. a video window for a videoconferencing system)

When an MMobject controls a multimedia data type (e.g. a video with stereo audio) the LINK command has the possibility to connect ports individually. For instance, connect the video, and connect the left audio channel to the mono incoming audio channel of the sink. Further considerations on this subject are out of the scope of this paper.

After the *Prepare* is returned (*Prepare* is the only action which has returning parameters) it contains reference information about the stream control interface (*port*) created by the BCO component via a control role to the binding object. This information is transferred within the *Connect* operation between the MMobject manager and the binding object manager. It will be used at engineering level to start the real connections to the different endpoints using the low-level *connect* (when it exists). The stream interface has a different reference identifier because ANSAware 4.1 does not support this interface and the effort to create a similar reference identifier was not considered relevant at this point. Nevertheless, this is not so important at conceptual level. Figure 4 shows the steps of the explicit binding (the *Connect* operation is

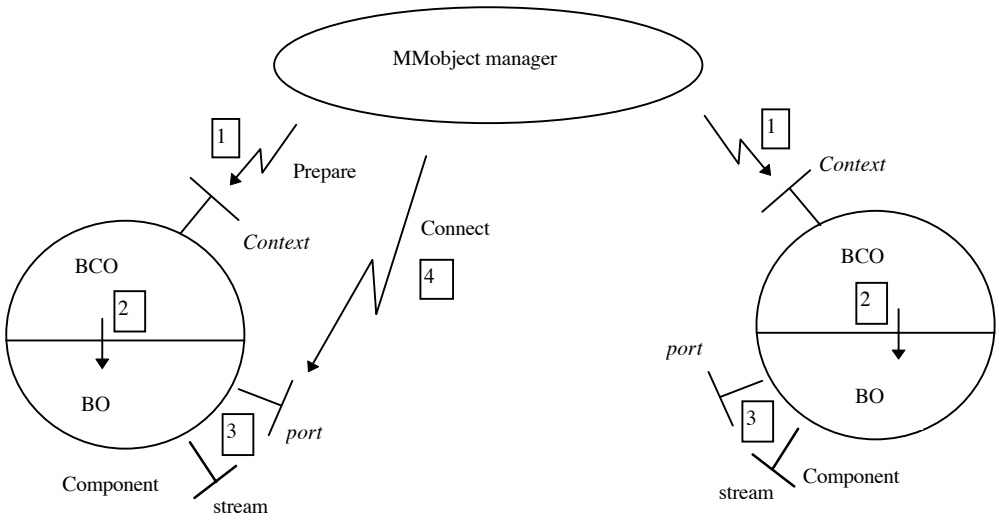

Fig. 4 - Computational steps for the explicit binding

invoked on the BO manager).

There are also control features related with the specific type of communication. An important one is the quality of service for the connection but it was not properly addressed in this version of the platform. BCOs can only set the size of packets and whether or not traffic shaping will be performed. Other parameters are hard-coded in the BO. Another control feature is the control of the floor when multipoint connections are managed. High-level policies are transferred using the control roles (only one gets the channel, volume is changed accordingly, etc.) and the BOs perform the respective algorithms.

4. Engineering Description

Basic computational objects and binding objects are supported by basic engineering objects and associated engineering concepts (stubs, channels, etc.). The operation interface part and the behaviour associated to the BCO use the ANSAware transparencies and functions, similar to the ODP standard. The novel part is the multimedia handling and it will be the focus of this section. The control part of the binding objects implementing streams is also straightforward because they use operation interfaces. The stream interface has the following correspondence at the engineering viewpoint.

A channel is created for the (possibly multipoint) stream connection inside the MMobject. The low-level *connect* operation (referred above for completeness of the description) is, in fact, an exchange of engineering interface references inside the channel. Each end point has a stub, which is specific to the *interface type* of the basic engineering object. Stubs perform little work because most of the multimedia data is handled by hardware devices controlled directly by the

basic computational object. The work consists of the identification of headers and in-band control information. The binder has a major role than stubs because it controls the multipoint connection. It implements the connection as a set of one to one connections in this version. The current implementation of groups in ANSAware was not seen appropriated to handle multimedia data due to great overheads. Implementing the binder with the awareness of 1:1 connections makes it ready to use multicast features of the ATM networks in the future (stronger arguments can be seen in the following section). Control information local to the node is added to the incoming stream regarding the control algorithms of the computational binding object described above (gain indications for the volume, etc.). The binder can also participate in the application if certain statuses related to its tasks are being considered (e.g. QoS). Finally, protocol objects are very lightweight because of the nature of the data. Different protocols are used to provide different quality of service assurances to the basic engineering objects.

Before establishing endpoints (when *Prepare* was called) a type compatibility check of the stream interface is performed. This check has two steps: the first is a protocol check and exists because stream references are not complete; the second refers to the encoding and high-level transport of data and is checked at run-time with the cooperation of the Type Repository.

The BCOs map onto basic engineering objects. They cooperate inside the MMobject to accomplish the actions offered to the exterior. They are also responsible to identify, process, and send events of interest to the application in general (other objects or application controllers). The two classes of events have different treatments. Deterministic event information is mixed with the raw data and sinks and sources deal with the event (depending on the type of the data, events, etc.). Note that there is no reason why stubs (or binders) could not interpret this kind of control information, except that the concept of what a binding object really is in the computational viewpoint would be more diffuse. Non-deterministic events are handled by sinks or sources and can come from the binder (decrease of QoS); from the user (clicks on windows); from the geometry manager; etc.

With respect to multimedia data and streams, sinks just pass the data from the stubs to the hardware devices (or another corresponding entity). Sources get the data from the producer (disk, microphone, camera, etc.) and feed the stub (actually the real-time preserving algorithms of the channel take initiative on the sampling instants making up-calls to the basic engineering object to transfer, or ignore, data).

5. Technological Description

The non multimedia part of the system uses the ANSAware 4.1. The stream interface was added to this package and consists of:

- Protocol objects are simply a direct use of UDP and TCP sockets in this version. A lightweight connection-oriented transport protocol, such as XTP, was felt to be missing. UDP takes a reasonable long time to process address information per datagram, and it is not possible to deactivate the flow control features of TCP. Direct use of the AAL5 interface to the ATM network would solve the problem. But it would also prevent this platform from running over different networks unless a major engineering work was performed to make the choice of a common protocol automatic.
- The binder manages a binding endpoint structure with all the connections to the other components. New link points are added each time there is a *Connect* and a low-level *connect* between binders. Multimedia data is sent once by the basic engineering object and transmitted to each endpoint. The binder relies on the protocol object to provide the necessary quality of service for the engineering object. Audio and video, for instance, use UDP transport without any error control. Some control information is provided for the

identification of the basic structures, such as a video frame. The binder accepts any frame as valid if the error rate (missing parts) is less than a threshold percentage. There is a mechanism for real-time rate control with awareness of missed deadlines which work with basic engineering object using up-calls.

- The stub performs unmarshalling of parameters avoiding memory copies as long as possible. Data is then copied into the hardware devices for decoding and presentation (a dual task is performed by client stubs).

It is important to refer that is this prototype the stream interface runs in ANSAware kernel mode (i.e. without any task context) to improve efficiency. Each time a deterministic interaction is necessary, the task context is searched before any operation interaction takes place. This creates a potentially unsafe part in the system at the expense of not creating efficient kernel structures to handle the stream interface. The problem is easily solved by integrating the binding endpoint structure within the kernel and schedule the stream activities before any other task. Once more, such modifications to the ANSAware were not considered relevant at this stage.

Basic engineering objects interact with each other using the ANSAware transparencies and functions to execute the dynamic behaviour of the application. The property feature of the Trader was used intensively to support the platform´s type system. Each time a new video, audio, text, etc. is created new instances of the objects are launched to handle it.

6. Using non-ODP systems

The application described in the Introduction (joint-synchronized access to an information system with videoconferencing) was implemented and it is the subject of the following section.

The videoconferencing application is the Berkom system and uses the XVideo board from Parallax for video and the workstation´s audio device. Both Berkom and the multimedia platform can coexist competing for the XVideo resource. Unfortunately, this is not true for the audio device. One system would have to integrate the other.

The audio type has the particularity that it is only the source which interacts at control level. Sinks simply present data. Therefore, the use of Berkom is just a communication issue because the dynamic behaviour of the application is not influenced. From the computational viewpoint a binding object would solve the problem...

The solution found was the definition of a special binding object (the ODP <x>interceptor was interpreted as an object to connect different ODP domains. The problem here was to access a non-ODP system). The new binding object still has the compliance to the *port* interface. Thus, at computational level nothing is changed except that a new MMobject must be used (one that uses the special binding object).

The subtyping of *port* is already an engineering viewpoint issue. The new subtype, called *mmcport*, has not any low-level *connect* operation defined and provides only a type definition. The Berkom system is accessed when the control role of the binding object is signaled by the basic engineering object (step 2 of figure 4). The *Connect* operation adds new destinations to the multipoint connection. This is performed by the Berkom system instead of calling the low-level *connect* as for the other binding objects described before.

There are still two aspects to be taken into consideration:

- differences on how to address endpoints (ODP references on DIMPLE versus machine names on Berkom);
  Engineering interface references for *port* interfaces are obtained by parameter passing, so the parameters actually exchanged were replaced. A selector decides if the reference is an ODP reference or any other alien engineering reference supported by DIMPLE. For the Berkom case the alien reference includes the name of the machine to connect to and the Berkom type for the data. The *mmcport Connect* operation maps directly into the appropriate data type of Berkom. Fig 5 shows the various standards involved when audio is played from the platform using Berkom.
- the acquaintance of the addresses (previously there was the *Prepare* operation on the BCOs and the addresses of the BCOs were obtained from the Trader).
  The Trader cannot provide interface references for the Berkom system, so the solution adopted was to get the address information from the application itself when a new connection for an MMobject is requested.

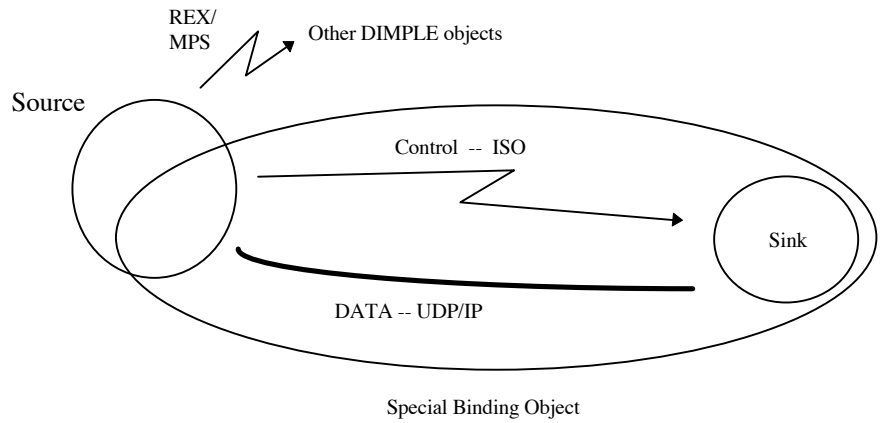

Fig 5 - Engineering and Technological view of the connection DIMPLE - Berkom

7. Example

The example application consists of a multimedia retrieval system with multimedia documents stored in a distributed way. The behaviour of the document is the real application on DIMPLE. MMobjects handle each type of data (video, audio, text, etc.) so the user can see the document at the sinks and interact with it (there are button objects and sinks are sensitive to mouse clicks). The specification states how MMobjects evolve to create the document and how and when users can interact. This application can be shared by simply connecting another set of sinks (representing a new user) to the active MMobjects. New users just get the current information existing users are watching. In control terms, the application is not aware that another user was added because the distributed MMobjects hide that fact. The binder controls the synchronization between user interventions internally. The videoconferencing system is the Berkom system, as already mentioned. It runs independently from the DIMPLE application but has to be running if it is going to be used during the session because of the audio device access. A more detailed description of this application can be seen in [5].

The specification is interpreted by a unique object in the system. Events are reported to it and actions are triggered from it. Decentralized control could also have been implemented and each MMobject manager would have an interpreter in it. This was done for other applications [4]. In order to keep track of who is in the system at every moment an object called UGM (User

Group Manager) keeps an array of users and an array of active MMobjects. This information is important because MMobjects have to receive "user connection" orders and the interpreter should not care about this. User connection orders are gathered in a yet another *status*, used exclusively by the UGM. The interpreter, the UGM and all the MMobject managers form a capsule called *sserver* (script server).

When a user wants to access the information system, (s)he runs a front-end application, called *smp* (shared multimedia presentation), with a name for the session -- let's assume castles. *smp* looks up the interpreter in the Trader, and invokes the *Prepare* action for an object of type *application* which instantiates the castles instance. The *Prepare* is intercepted by the UGM and an exclusive *Context* status interface is given to the *smp* (see figure 6). *Prepare* has the node name of the user's machine that is used to filter offers in the Trader and to identify Berkom sinks. The interpreter eventually receives the *Prepare* invocation deliver and invokes the *Prepare* on all MMobjects of the document. Source components are running and sink components are produced via the factory object. Meanwhile, a window control interface appears on the user's screen (see figure 7), giving the user the possibility to play, stop, pause and quit the application.

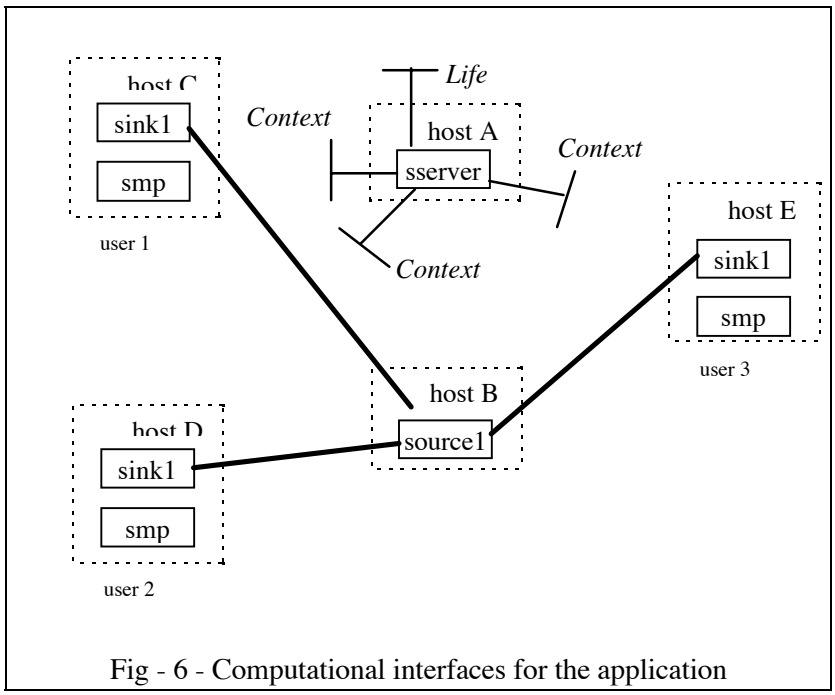

Fig - 6 - Computational interfaces for the application

When another user wants to join the session he must know the session name. His *smp* proceeds exactly as the first one but this time UGM traps the *Prepare* and does not let it go to the Interpreter. Instead, it invokes the *AddUser* operation on the MMobjects "user connection" status. Each MMobject reacts to this operation according to the replication level of its components, creating the necessary sinks and linking to the necessary sources. It also creates another exclusive *Context* status interface for this new user and another window control interface is presented.

The existence of different instances of the *Context* status interface is simply an easy way to control the *smp* accesses. The Interpreter receives the deliver of, for instance, the *Play* invocation regardless of what *smp* has submitted it.

When a user quits *smp*, the *Destroy* operation on his *Context* status is invoked. Once more the UGM intercepts the invocation and if the user is not the last one prevents the call to follow to the Interpreter. The *RemoveUser* of the "user connection" status of the MMobjects is invoked and the connections related to this user are destroyed. When the last user submits a *Destroy*, the Interpreter gets it and destroys the application instance.

Figure 7 shows the output of the application with the window control interface, a picture and a movie. Other users see exactly the same thing.

Fig - 7 - An example of a document with a picture and a video about castles

There are two interesting remarks to make. First, note that the MMobject manager is exactly the same for any type of object -- it works at status level and is completely parameterized by the status, property list, component definition, etc. with the run-time help of the Type Repository. At its level of abstraction there is no difference between a movie, an audio or a picture MMobject. In fact, the example application has only one manager inside the *sserver* for all MMobjects. For the second remark it is necessary to say that any object can send an *Ev_Start* event when it starts and can send an *Ev_End* when it finishes (to any objects that showed interest). These events are related to the status *Context*. Note now that the application is seen by the *smp* via a *Context* status. All *smps* can receive events *Ev_Start* and *Ev_End* each time the Interpreter starts the playing of the object or finishes. This means that an application is just another MMobject that can be included in any higher-level specification, and so on.

8. Related Work

Several works describe experiences covering certain parts of what was discussed in this paper. The ones which overlap the most are: [11] where the general concept of a multimedia socket and plug is defined with a reduced set of actions and some type compatibility rules; [7] contains the five viewpoint specifications of what a stream binding object should be. However, most of the functions are actually considered at the level of basic computational object and engineering objects such as the binder and the stub were omitted. It was seen in DIMPLE how important their roles are. [7] also uses TCP for data transportation which can prevent any practical application due to the retransmission algorithm. [6] presents a general taxonomy for the CSCW in open distributed systems but does not cover continuous media very deeply.

9. Conclusions and Further Work

This paper has shown how an application for multimedia cooperative work can be easily constructed with the concept of distributed objects and encapsulation of the control of data. The DIMPLE platform works at control level which is the level required for most of the general purpose multimedia applications. It is difficult, of course, to express data manipulation at this level (e.g. particular algorithms for mixing signals) but the simplicity gained in constructing other types of applications overcome this limitation.

A key concept for the distributed objects defined here is the binding object from the RM-ODP, and specially its relation with the stream interface. This is a subject not fully addressed on the standard and the experience gained with the work of this paper was elucidating in some aspects. A first observation is the need for operation interfaces to be able to control the binding object from other computational objects. However, other operations are needed to fully control the binding from an engineering viewpoint. Subtyping was chosen here as the solution. As a second observation, stream interfaces need explicit bindings because of the different nature towards their operation counterparts. The paper has shown how intuitive it is to drive the binding from constructs in the language.

At engineering level most of the work of the channel is performed by the binder, and its interface needs to be carefully defined in order not to restrict any function related with multimedia data. Stubs only look at control information to introduce as little overhead as possible.

At technological level it was felt the need for multimedia transport protocols and clever usage of the transmission medium by the binder. The high-performance connection of the stub to the hardware devices is also a critical point in the overall efficiency.

The binding object concept models communication in such a good way that it can hide limited scale interworking with non-ODP systems from the computational viewpoint. Manipulations at engineering level were sufficient.

Some issues are still open for further research. The binder interface needs a clearer definition of high-level QoS parameters for the data streams, and the definition of the proper concepts to exercise control of the binder features. It is also an open question whether the binder could actively participate on the DIMPLE control narrowing the space for the basic computational objects in the components.

In terms of protocol objects, a low-level synchronization algorithm to enforce intra-stream synchronization and inter-stream synchronization between different media in an MMobject is currently being developed and will be reported in the near future. The current real-time enforcing algorithm has no control on the continuity of data.

A final point is the definition of a syntax to express complex topologies in the language. Some limitations were felt with the current syntax specially for odd cases where hardware devices force a certain configuration (for instance, the XVideo board insists on having a video window on the source, making the need for a local sink null).

Acknowledgments

The authors wish to acknowledge Companhia Portuguesa Rádio Marconi for the partial funding of this research. Part of this work was performed under the scope of an EURESCOM project called EMMA - *European Multimedia Experiments in an ATM Environment*.

References

[1] M. Altenhofen, J. Dittrich, R. Hammerschmidt, T. Käppner, C. Kruschel, A. Kückes, T. Steinig, The BERKOM Multimedia Collaboration Service, ACM Multimedia 93, pp. 457-463.

[2] D. P. Anderson and G. Homsy, A Continous Media I/O Server and Its Synchronization Mechanism, IEEE Computer, Special Issue on Multimedia Information Systems, 24(10), pp. 51-57, 1991.

[3] ANSAware 4.1, System Programming in ANSAware. Document RM.101.02, February 1993.

[4] L. Bernardo, Specification and Synchronization of Multimedia Aplications with Distributed Control, Msc Thesis, Instituto Superior Técnico, Lisboa, Portugal, June 1994 (in Portuguese).

[5] L. Bernardo, P. Pinto, Sharing Multimedia Information: a basis for assisted remote training, Proceedings BRIS'95, Dublin, September 1995.

[6] G. Blair and T. Rodden, The Challenges of CSCW for Open Distributed Processing, International Conference on Open Distributed Processing 1993, pp. 99-112.

[7] V. Gay, P. Leydekkers and R. Veld, Specification of Audio/Video Exchange Based on the Reference Model of ODP, Proceedings BRIS'94, pp. 179-191.

[8] ISO 10744, Information Technology - Hypermedia Time-based Structuring Language (HyTime), 1992.

[9] ISO/IEC 10746-3, ITU-T Rec. X.903, Open Distributed Processing Reference Model - Part 3: Architecture, 1995.

[10] ISO 13522-1, CD. Information Technology - Coded Representation of Multimedia and Hypermedia Information Objects (MHEG) - Part1: Base Notation (ASN.1), 15 June 1993.

[11] C. Nicolaou, An architecture for real-time multimedia communication system, IEEE Journal on Selected Areas in Communication, 8(3), pp. 391-400 (1990).

[12] P. F. Pinto, P. F. Linington, A language for the specification of interactive and distributed multimedia applications, International Conference on Open Distributed Processing 1993, pp. 217-234.

[13] P. Pinto, L. Bernardo, P. Pereira, A Constructive Type Schema for Distributed Multimedia Applications, Proceedings BRIS'94, pp. 419-434.

[14] S. Ramanathan and P. Venkat Rangan, Feedback Techniques for Intra-Media Continuity and Inter-Media Synchronization in Distributed Multimedia Systems, The Computer Journal, 36(1), pp. 19-31, 1993.

[15] J.Rodríguez, ISABEL: Quick Reference, Dpt. Ingeniería Telemática, E.T.S.I. de Telecomunicacíon, Univ. Politécnica de Madrid, December 1994.

[16] J. Stefani, L. Hazard, and F. Horn, Computational model for distributed multimedia applications based on a synchronous programming language. Computer Communications, 15(2), pp.114-128, March 1992.