A Multimedia Application Programming Interface Framework

Max Roger Pokam
LGI/IMAG (Grenoble)
e-mail: Max.Pokam@imag.fr
Gérard Michel
LGI/IMAG (Grenoble)
e-mail: Gerard.Michel@imag.fr

Abstract
Along with the emergence of high-speed communication technologies and protocols, tools for multimedia applications are growing in terms of their variety and their performances, making multimedia data flows candidates to the integration in almost every computer system. We particularly focus on the case of the factory plant where audio and video equipments could soon take part in the control/monitoring of manufacturing processes. In this paper we propose a MMS-like multimedia application interface for such an environment.

Keywords: Application Programming Interface, Multimedia, Distributed Systems, Application layer, Computer Integrated Manufacturing, MMS.

1 Introduction
With the advent of high speed network technologies (gigabit networking [8]), powerful computer architectures with processors achieving tens of MIPS and high speed bus interfaces like the futurebu+ [4], it is our believe that audio and image traffics will soon be included in almost every information processing system. We are particularly interested in what will become of the common manufacturing floor with it’s rather time-critical applications. Some examples of gigabit applications in the manufacturing systems are the tele surveillance and the Journal management.

The tele surveillance application in the manufacturing environment provides means for the survey of remote sites. Remote cameras and microphones are used to capture image and audio data that are sent to displays, loudspeakers, storage systems, or to specific sig- nal processing systems. A tele surveillance application as defined here is similar to the telerobotics application mentioned in [7]. The journal management application, on the other hand, provides means to record time-stamped image and audio information related to specific events that could occur on the manufacturing plant.

With the progress being accomplished in computing technologies, the mixing of continuous media streams to other information streams on the manufacturing floor is made possible, resulting in a wider significance of the word Integrated in the phrase "Computer Integrated Manufacturing (CIM)"). Multimedia and manufacturing communication systems will not be implemented separately. Rather, they’re going to be combined in order to share the same processing and networking resources. The aim of our paper is the de-

Figure 1: The Application Layer

Proceedings of the Fourth International Conference on Database Systems for Advanced Applications (DASFAA'95)
Ed. Tck Wang Ling and YoshihumiMasunaga
Singapore, April 10-13, 1995
© World Scientific Publishing Co. Pte Ltd
ment (ASE) of the application layer in the OSI reference model. The meaning of "objet" in this approach is slightly different from the one in conventional object-oriented paradigm[6] in which an object is a combination of attributes and methods, so-called encapsulation. An "object" in our model is a method with attributes, called associated action.

The major entities of the derived Application Service Element are sketched on figure 1. The description of the objects and services used for their management, as considered in our model of distributed multimedia system integration in a manufacturing system, is given in the remainder of the paper. Section two is a presentation of the virtual multimedia device (vmd) with the related objects, attributes and services. Section three is devoted to the service agent (sa), an entity which main role is the realization of the services requested by client application programs. Section four describes the environment management agent (ema) which is involved in every service affecting the distributed multimedia environment. Section five presents the environment information base (eib), which contains all the information on the multimedia environment. It is accessed by the environment management agent. Finally, section six concludes the paper.

2 The Virtual Multimedia Device

The virtual multimedia device is just like its counterpart, the virtual manufacturing device of the manufacturing message specification. It is an interface to real devices capable of mapping real objects of the equipment onto virtual ones, and making them available to client application programs through the use of specific services. More precisely, the virtual multimedia device, the vmd, interfaces equipments that restore, store, or generate continuous audio and image data flows. The specification of the vmd object is given on figure 2. The key-attribute to identify the vmd throughout the system is its vmd ID. The Type attribute allows the distinction among AUDIO-SOURCE, AUDIO-SINK, IMAGE-SOURCE and IMAGE-SINK devices to be made. The State attribute tells whether the device is operational or if it needs some commissioning. The List Of Coding Methods Supported attribute contains the list of audio or image information formats the device is capable of handling. The List Of vmd Objects attribute contains the list of all other objects of the vmd.

The basic operations that can be performed on a vmd object are: the Identify operation used to get the vmd ID attribute of the vmd, the Status operation used to get the State attribute of the vmd, and the GetVmdAttributes operation used to get all the attributes of the vmd object.

The principal objects of the vmd are the continuous media agent cma, the digital signal processor agent dspa, the semaphore, the event, the stream, and the journal. Their descriptions are given in the following subsections.

2.1 The continuous media agent object (cma)

The continuous media agent is the entity responsible for the transmission or the reception of continuous data flows. The role of the cma agent is to translate data flows from the multimedia device directly into transport PDUs when the device is a source, or to directly translate transport PDUs into data flows delivered to the multimedia device in the case of a sink device. It also synchronizes the exchanges between the transport connection and the multimedia device.

The specification of the cma object type is given on figure 3. The cma object has a key-attribute, the cma ID attribute, which identifies the object throughout the communication environment. The MaxBitRate attribute gives the maximum bit rate at which the continuous media agent can operate when transmitting (when the device is a source) or receiving (when the device is a sink) data flows.

The operations that are performed on the cma object are: the GetCmaAttributes operation used to get the current attributes of the cma object, the Start operation used to start a cma object from the IDLE state, bringing it into the RUNNING state, and the Stop operation used to stop a cma object that is in the RUNNING state and bring it in the IDLE state.
2.2 The digital signal processor agent object (dspa)

In distributed multimedia applications there is often a need to perform a signal processing algorithm over a piece of data in order to detect relevant events and initiate corresponding actions. The digital signal processing agent of the vmd serves this purpose of signal processing. The specification of the dspa object type is given on figure 4. The dspa is identified by a key-

| Object: dspa |
| Key-Attribute: 0 ID |
| Attribute: Bandwidth |
| Attribute: State (IDLE, RUNNING) |

Figure 4: The dspa object

attribute, the Algorithm ID, which is the identifier of the processing algorithm it provides. The Bandwidth attribute is an integer giving the number of data units that the dspa object can process in a unit of time. The operations performed on the dspa object are identical to those described above for the cma object.

2.3 The semaphore object

The classical problem of access control arises when more than one client application programs request a shared resource like the dspa. A semaphore is used to protect the shared resource and to control its allocation policy. In our model, requests are prioritized and queued. The preemption is governed by the priority ceiling protocol. The specification of the dspa object type is given on figure 5. The Shared Resource

| Object: Semaphore |
| Key-Attribute: Shared Resource ID |
| Attribute: Owner ID |
| Attribute: List Of Requesters |
| Attribute: Priority |

Figure 5: The semaphore object

ID is the key-attribute that identifies the semaphore throughout the communication system. The Owner ID attribute is the identifier of the present owner of the semaphore; a NULL value of this attribute will indicate that the semaphore is free at that moment. The List Of Requesters attribute is the list of client application programs that are waiting to be granted the shared resource by the semaphore. As the result of using the priority ceiling protocol, the Priority attribute gives the highest priority of semaphore entries which priorities are higher than the priority of the present owner.

The operations that can be performed upon a semaphore object are: the GetSemaphoreAttributes operation used to get the various attributes of a given semaphore object, the RequestSemaphore operation used to request the control of a shared resource, and the RelinquishSemaphore operation used to free a shared resource and make it available to the requesting client application program having the highest priority.

2.4 The event object

An event condition is made when the results of the computations of the dspa are coherent with a target (e.g. the presence of a specific item is detected in a scene, this corresponds to an event object in the model). An associated action is then carried out. The specification of the event object is presented on figure 6. The Event Condition Name attribute is the key-

| Object: event |
| Key-Attribute: Event Condition Name |
| Attribute: Event Action |
| Attribute: List Of Modifier |

Figure 6: The event object

attribute identifying the unique condition on which the event will be detected. The Event Action attribute is the action due to take place whenever the event occurs. The List Of Modifier attribute gives all the conditions that should be verified prior to the event action. The Application Program attribute identifies the client application program on behalf of which the event object is being managed.

The operations performed on an event object are: the CreateEvent operation that allows a client application program to create an event (a prerequisite to this operation is to obtain the control of the dspa semaphore), the DeleteEvent operation that allows a client application program to destroy a previously created event object, and the GetEventAttributes operation that returns the values of the attributes of a specified event object.

2.5 The journal object

The journal object of the vmd contains time-stamped sequences of audio or image data recorded at the occurrence of particular events. The specification of the journal object is presented on figure 7. The Journal Name attribute identifies the journal object throughout the communication environment. The List Of
Figure 7: The journal and document objects

**Documents** attribute gives the list of documents recorded and present in the journal object. The operations that can be performed on a journal object are: the **CreateDocument** operation that allows a client application program to create a new journal entry. This operation can also be the action to carry out at the occurrence of an event, the **DeleteDocument** operation that allows a client application program to delete an entry of the journal object, and the **GetDocument** operation used by a client application program to download a particular document from the virtual multimedia device. For the two last operations, the access rights to the document should be checked before they could be performed.

### 3 The Service Agent (sa)

The model of distributed multimedia system we propose is client-server oriented. The operations on **vmd** objects and those on environment objects presented later in sections 5 and 4 are accomplished through the exchange of **service primitives** as shown on figure 8. Those primitives are transmitted over an **association** that needs be established between a client application program and a server (**vmd**). There is a service agent at each edge of the association to monitor the realization of the service from beginning to completion.

![Client-server model](image)

**Figure 8:** The client-server model

The service agent receives a **request** service primitive from the application program and translates it into a transport protocol data unit, a **RequestPDU**, which is transmitted over the transport service access point (**T-SAP** on figure 1).

At the **vmd** side, the service agent receives a **RequestPDU** from the transport service access point and translates it into an **indication** service primitive it delivers to the **vmd**. At the reception of the indication service primitive, the **vmd** performs the corresponding operations on its objects (if needed) and returns a positive or negative **response** service primitive with relevant arguments to the service agent. The latter translates the response service primitive into a **ResponsePDU** if the response is positive, or into a **ErrorPDU** in the case of a negative response. The service agent then transmits the **ResponsePDU** or the **ErrorPDU** protocol data units over the transport service access point of the association.

At the client side, the service agent will receive the **ResponsePDU** or the **ErrorPDU** protocol data units and they will be translated into a positive or negative **confirm** service primitive respectively and be delivered to the client application program.

All services do not have primitives that are directly translated into transport protocol data units. Services related to the communication environment, like the **Initiate** service which establishes an association, require the intervention of the environment management agent (**ema**) to be completed. This is the main reason why the **ema** agent is represented in a dashed-line box on the path from the **sa** agent to the transport service access point on figure 1. In such a case, the **ema** agent receives the request primitive, performs a transaction on the environment information base (**eib**), transmits a transport PDU to a peer **ema** agent. At the reception of the PDU, the peer **ema** agent also performs a transaction on the **eib** database, translates the PDU into an indication service primitive delivered to the **sa** agent that finally delivers it to the server **vmd**.

![Sa finite state machine](image)

**Figure 9:** The **sa** finite state machine

Figure 9 shows the four states of the **sa** finite state machine: the **idle** state in which nothing is being done, the **tsap interaction** in which the **sa** agent is interacting with the transport service access point, the **ema interaction** state in which the **sa** agent is interact-
ing with the *emu* agent, and the **appli interaction** state in which the *su* is interacting with the client application program or the server *vmd*. The tree last states are timeout-controlled and state transitions are based on the reception of service primitives or PDUs (serv.req <-, RequestPDU <-) or on the emission of such data units (ser.req ->, ResponsePDU ->).

### 4 The Environment Information Base (*eib*)

The environment information base is a database containing all the information needed in the management of the distributed multimedia system. Each of the following subsections is devoted to the description of a particular environment object whose attributes are maintained in the *eib*.

#### 4.1 The association

The *association* is often defined as the connection at the application level. The *association* is the support of the *control port* as shown on figure 1. It is a full-duplex connection over which service primitives are reliably exchanged between a client application program and a server *vmd*. The specification of an *association* object is given on figure 10. The *association* object is identified by an *Association ID* attribute. The communicating application entities are identified by the *Calling Party ID* attribute for the party that created the association, and the *Called Party ID* for the party that accepted the creation of the association. The *Calling Control Port ID* and the *Called Control Port ID* attributes respectively identify the control ports the calling and called application entities will later use to transmit and receive each its service primitives. The *Calling Bound T-SAP* and the *Called Bound T-SAP* attributes contain the transport service access point identifiers allocated at both edges of the association. The *Quality Of Service Parameters* attribute contains the traffic and service requirements put upon the underlying transport connections and service agents. It describes requirements such as bit rate, end-to-end delay and the maximum number of outstanding services.

The operations that can be performed on an *association* object are: the *Initiate* operation requested by a client application program when it needs an *association* with a server *vmd* to be setup, the *Conclude* operation that can be requested by either of the communicating entities to tear down the *association*, the *GetAssociationAttributes* operation that can be requested by an application program to get the attributes of the *association*, and the *BindToTSAP* operation used when one of the transport service access point of the *association* is to be changed.

#### 4.2 The channel

As mentioned in the previous subsection, the *association* is use for the transfer of service primitives. The transfer of real-time continuous data flows between a source *vmd* and a set of sink *vmds* is provided by dedicated connections known as *channels* in our system. The specification of a *channel* object is given on figure 11. The *channel* object is intrinsically related to the source that generates data flows over it. The *Source ID* is then the key-attribute of the *channel* object. The *List Of Sink IDs* attribute gives the identifiers of all the sink devices receiving the data flow transmitted over the *channel*. The *List Of Sink IDs* attribute will contain a unique identifier in the case of a unicast operation. The stream port (see figure 1) is equivalent to the control port allocated to the association. A stream port is used by a *cma* agent to receive/deliver data flows from/to the real device. For a *channel*, there is a source stream port identified by the *Source Stream Port ID* attribute and one or more sink stream ports identified by the *List Of Sink Stream Port IDs*.
attribute. Similarly, a channel is allocated a source T-SAP identified by the Source Bound T-SAP ID attribute and one or more sink T-SAPs of the List Of Sink Bound T-SAP IDs attribute.

The operations that can be performed on a channel object are: the CreateChannel operation used to create a new channel the AddToChannel operation used to add a new T-SAP to the list of receiving T-SAPs, the RemoveFromChannel operation used to remove one T-SAP from the list of receiving T-SAPs, the DeleteChannel operation used to completely destroy the channel object, and the GetChannelAttributes operation used to return all the attributes of the channel object.

4.3 The application entity identifier (aei)

When a new application program or vmd enters the system, the action prior to every further interaction with the system is to be registered and be allocated an identifier. The specification of the application entity identifier object is given on figure 12. The two operations affecting the aei object are: the Register operation used to create a new entry of aei object in the eib database, and the Unregister operation used to delete an entry of aei object from the eib database.

5 The Environment Management Agent (ema)

The environment management agent is used when the requested operations need at least one transaction on the environment information base eib. Figure 13 shows the four states of the ema finite state machine: the idle state where nothing is being done, the tsap interaction in which the ema agent is interacting with the transport service access point, the sa interaction state in which the ema agent is interacting with the sa agent, and the eib transaction state in which the ema is performing a transaction with the eib database. The three last states are timeout-controlled and state transitions are based on the reception of service primitives, transaction responses or transport PDUs (serv.req <-, RequestPDU <-, trans.rsp <-) or on the emission of such service primitives or transport PDUs (ser.req ->, ResponsePDU ->).

6 Conclusion

The client-server and object-oriented MMS-like model we specified for a multimedia application programming interface present multimedia devices (source and sink) and the channels connecting them as virtual objects on which client issued command and control service requests can be performed. It provides means to control multimedia devices via the vmd object and other related objects, to start or stop special signal processing algorithms through the dspu object and other related objects, to start or stop special signal processing algorithms through the event object, to record audio and image documents through the journal object, and to manage the channel object carrying audio and image data flows. As far as the implementation of our model is concerned, it is aimed to be a meta-kernel added to any existing real-time operating system kernel providing support to multimedia devices and services. The issues of interprocess communication, process scheduling, context switching and interrupt handling are of major importance in the choice of the supporting real-time OS kernel. We are looking forward using a micro-kernel system architecture, because of its flexibility (QNX and Chorus are possible choices), as the supporting micro-kernel operating system.

References


