A Continuous Media Orchestration Service: Draft Contribution to Discussion on the ISO NWI on Enhanced Transport Mechanisms

ESPRIT project OSI 95 contribution: edited by Lancaster University, UK and Alcatel ELIN, Austria[1]

January 1992

ABSTRACT

Standards have an important role to play in development of multimedia. It is important for the commercialisation of multimedia that standards are agreed for the representation and interchange of multimedia. It is also important that existing standards are adopted where possible to allow multimedia to be integrated into the current generation of systems. However, currently OSI standards do not take into account the particular requirements of high performance multimedia applications, for example, the OSI reference model does not support the kind of multimedia synchronisation required by continuous media applications. This paper describes proposed new ***orchestration*** *services that meet the needs of distributed multimedia applications surveyed in a recent study carried out in the ESPRIT OSI 95 project.*

1. Scope

This paper details a new set of high speed transport and *orchestration* services required to support isochronous multimedia traffic over HSLANs and B-ISDN. These continuous media services include connection oriented, data transfer, quality of services negotiation and re-negotiation, and orchestration services. In particular, we concentrate on the need for co-ordination, or orchestration, of multiple related continuous media streams. The areas of group/multicast addressing and high speed transport protocols are not addressed in detail, but dealt with briefly in the final section of the paper.

The intention of this paper to introduce the results of our work in the area of distributed multimedia systems. Although our service specifications are expressed in OSI-like terminology, neither the terminology or the specifications themselves are to be taken as definitive. Our purpose is to stimulate discussion within the standards bodies rather than to promote mature proposals.

2. Introduction

The evolution of multimedia computing is being influenced by new application areas and by the increasing capability of high-performance multiservice networks. Multiservice communication technology is evolving rapidly, particularly in high bandwidth MANs, such as DQDB and FDDI-II, and B-ISDN based on fast packet switching principles. Currently, broadband transport mechanisms such as ATM are capable of supporting a wide range of multimedia platforms with diverse QoS requirements, from low speed voice (32 Kbit/s) transmission, to very high speed HDTV (100-150 Mbit/s) distribution between subscribers.

There are many application areas where the need for distributed multimedia capabilities is central. A recent survey [Williams,91] characterises such application areas as: office automation, service industry applications, retail applications, domestic applications, science and engineering, and cultural activities. The single most significant attribute of such applications is the incorporation of continuous media, particularly audio and video, although the transmission of other data types such as image, text and graphics will normally be involved. The significance of continuous media communications lies in the real-time, isochronous nature of the traffic which is not adequately supported by existing communications sub-systems [Blair,92].

---

[1]E.mail: (Lancaster) mpg@comp.lancs.ac.uk ; (Alcatel ELIN) lan_leop%rcvie@net.eu.austria

UK Expert Contribution to the ISO New Work Item on Enhanced Transport Mechanisms, Paris, February 1992. Referenced under BSI index: BSI/IST 6/-/2/718

This paper is structured as follows. We first describe, in section 3, a set of requirements that are used to determine the communications support for distributed multimedia applications. Then we introduce, in section 4, our proposed continuous media transport and orchestration service. This is followed by a discussion of other multimedia considerations in section 5, and finally a summary in section 6.

It is important to stress that this proposed contribution deals with the additional communications **service** requirements that emerge from consideration of **continuous media**, and that other issues such as groups/multicast are for further study. Neither does this paper cover details of the associated protocol specification: this is being addressed within the ESPRIT OSI 95 project and is the responsibility of the University of Liège.

3. Requirements study

The proposal contained in this paper result from work carried out by the OSI 95 project. The aim of the Lancaster contribution to this project is to take a top down approach to determining requirements for OSI protocols by studying distributed multimedia applications. This complements the work of other partners who are considering the impact of new communications techniques, particularly ATM/B-ISDN. This overall strategy is illustrated in Figure 1.

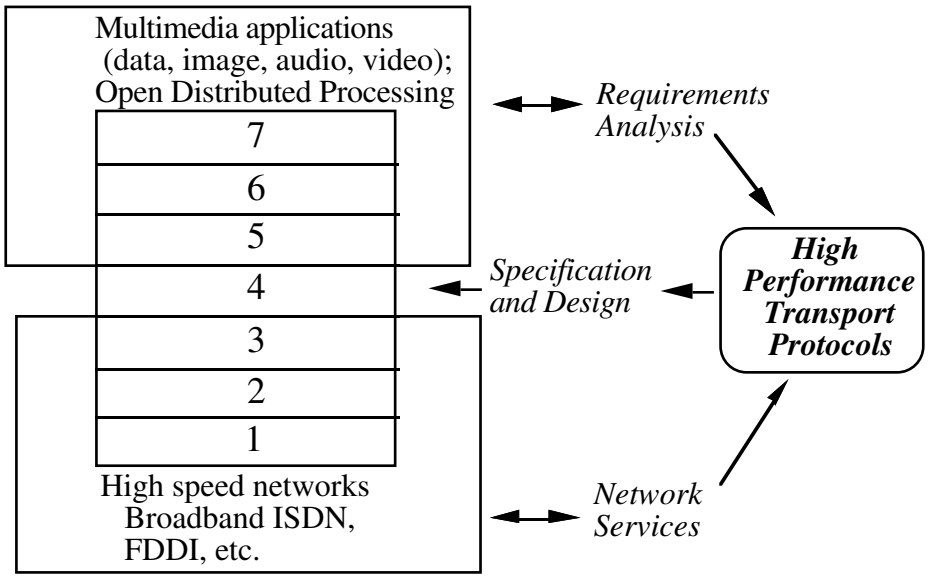

Figure 1: OSI 95 overview

3.1 State of the art in applications

There are a number of application areas where the introduction of multimedia can enhance the utility of existing systems. In addition, there are many new areas of application made possible with the emergence of multimedia, including office automation, service industry applications, retail applications, domestic applications, science and engineering and cultural activities. The topic of Computer Supported Cooperative Work (CSCW) is also highlighted as an important development with implications for distributed multimedia technology.

In total, the application survey covers over 100 projects including European projects funded by ESPRIT, RACE, DELTA and AIM, as well as other projects in Europe and the United States. The overall balance of projects is shown in Figure 2.

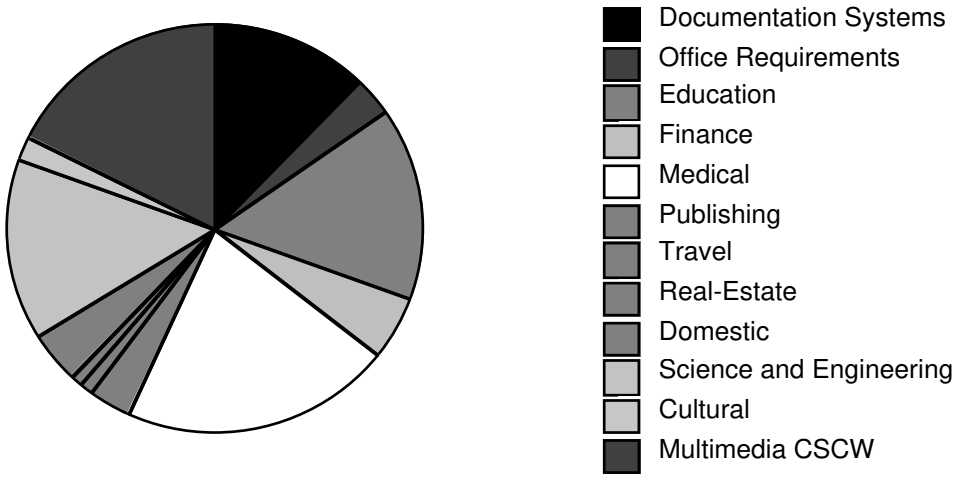

Figure 2: Areas of Application for Distributed Multimedia Computing

From this survey, the following general comments can be made about the state of the art in distributed multimedia applications:-

i) interest in distributed multimedia applications

There is a great deal of interest in distributed multimedia applications, with a surprising level of activity uncovered in European programs such as RACE, ESPRIT, DELTA and AIM as well as other countries.

ii) diversity of application areas

Distributed multimedia computing has applications in a wide range of areas, such as the office environment, health, the financial sector, teaching, science/ engineering and in cultural activities.

iii) importance of integration

Integration is the biggest motivating factor in most multimedia applications. More specifically, the majority of applications are concerned with the integration of a variety of information systems into a common framework.

iv) lack of commercial involvement

At present, most distributed multimedia applications are at an experimental or research and development stage. Commercial products in multimedia seem a long way away. The only area where products are emerging is in the special case of stand-alone applications, i.e. multimedia applications for single workstations.

v) ad-hoc solutions

Many of the applications listed in this report have been developed on either analogue technology or on a hybrid analogue/ digital solution. Few have experimented with completely digital solutions. This has allowed interesting prototypes to be developed but has deferred many of the technical problems associated with digital media.

vi) importance of the end user

A number of previous experiments with multimedia have failed (most notably videophones) because the technology has not been popular with end users. It is therefore important, in future designs of distributed multimedia applications, to take into account the needs and wishes of end users and to involve them in the design process. This requirement is illustrated by the growth of the field of CSCW in recent years.

3.2 Discussion of issues

In the context of OSI 95, it is important to appreciate the impact that the developing distributed multimedia applications will have on the OSI protocols. The new high performance protocols must support the multimedia applications requirements such as the transport of high-speed media traffic. This section highlights the major issues raised in section 2, the multimedia and distributed processing application studies, and categorises these issues as general protocol requirements, Quality of Service (QoS), orchestration and groups/multicast support. The major issues are detailed below:-

General

• ***support for a variety of services*** **-** It is important for the communications infrastructure to support a number of styles of service. The following are seen as particularly important for multimedia: efficient *transactions* (i.e. remote procedure calls) and high throughput *streams*. In multimedia systems, transactions are often used for control data whereas streams are used for continuous media traffic.

• ***profile selection*** **-** The concept of selecting a profile through the protocol stack is likely to be important in multimedia. Given the varying demands on a protocol stack by multimedia traffic, it is important to be able to select the appropriate path through the stack to provide a given service.

• ***end to end argument*** **-** In multimedia, it is not sufficient to guarantee the transmission of information between two endpoints. It is also important that the operating system can provide appropriate response. Hence, QoS issues such as throughput, latency and real-time responsiveness are end to end concerns as well as concerns of the communications infrastructure. This makes the relationship between the operating system and the communications infrastructure particularly important in distributed multimedia systems.

Quality of Service (QoS)

• ***QoS parameterisation*** **-** QoS is one of the most important issues in multimedia. The requirements of different media types vary greatly (e.g. voice and video) Similarly, a particular media type can be transmitted with a variety of characteristics (black and white image or full colour image). It is therefore important to support a notion of QoS whereby the precise characteristics of a transmission can be established. Again, this applies to both transactions and streams. Important QoS parameters for transactions are the *semantics* of the transaction (at most once, at least once, etc) and the style of synchronisation (synchronous or asynchronous). Important QoS parameters for streams include *throughput*, *error rates*, *latency* and *jitter*.

• ***QoS negotiation and re-negotiation*** **-** It is important that QoS parameters can be negotiated, i.e. it should be possible to interactively determine the QoS depending on the current state of the network, the volume of traffic, etc. Similarly, because of the dynamic nature of communications systems, it is also important to support re-negotiation of the parameters, e.g. to reduce latency as more bandwidth becomes available.

• ***real-time responsiveness*** **-** The real-time responsiveness of the communications system is particularly important for multimedia. i.e. the timeliness of delivery. This can actually be considered as a QoS issue but is worth highlighting in its own right. Responsiveness is important for transactions as real-time control environments demand guarantees of the temporal behaviour of control messages. This is also likely to be an important feature for control in multimedia environments. Similarly, timeliness is an important issue for stream services as media types such as audio and video require that the temporal structure of the information should be retained.

Orchestration

• ***multimedia synchronisation*** **-** There is a well known need in multimedia to support *continuous synchronisation* across a number of streams, e.g. lip-sync between audio and

video. In order to achieve this level of synchronisation, it is necessary to have a management entity monitoring the performance of underlying protocols and responding to exceptions or changes in the network. This introduces the notion of an *orchestrator* which oversees a number of stream connections. Note that orchestration requires the ability to respond to exceptions such as the temporary loss of synchronisation. This strengthens the argument above to support dynamic re-negotiation of QoS parameters.

Groups and multicast support

- ***groups and multicast support -*** The applications study highlights the importance of multicast services for multimedia. For example, many CSCW applications require sophisticated patterns of connectivity between different sites. Note that multicast is important for both transactions and streams.

All of these points presented above will have a significant impact on the OSI seven layer model, furthermore, these principal issues will be used to determine the specification of the new transport protocol to support distributed multimedia applications for high speed networks.

3.3 Other studies/proposals

Note that a number of other projects have examined the requirements imposed by multimedia on transport services, most notably an ANSI X3S3 report [ANSI,91]. We believe that the list presented above subsumes all the issues raised in the ANSI report. Within the EC RACE programme, project R.1071 [RACE,91] categorises the applications into a number of service domains. More recently, ANSI X3S3.3 has produced proposals for a new High Speed Transport Service (HSTS) and Protocol (HSTP), and these will be further studied in OSI 95. However, a preliminary assessment indicates that these proposals may complement the proposal for orchestration services contained in this paper.

4. Recommendations on continuous media transport and orchestration service

The continuous media transport and orchestration service outlined in this section is based on the model illustrated in Figure 3.

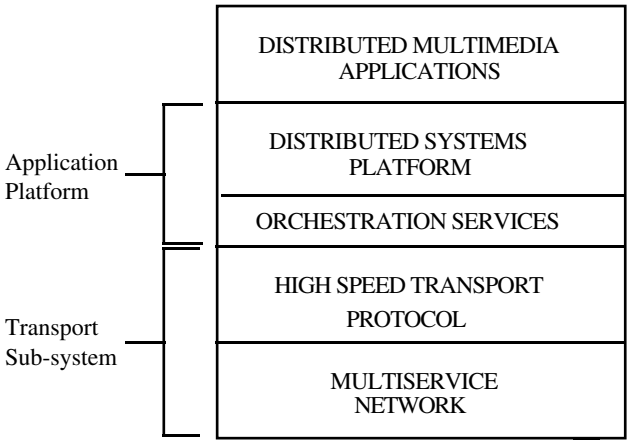

Figure 3: Distributed multimedia architecture

4.1 Continuous Media Transport Service

Connection Management Services

Connection management services, which are detailed in the following sub-sections,, which are detailed in the following sub-sections, deal with the establishment, release, and management of transport VCs.

Remote connection

In computer supported cooperative work (CSCW) and multimedia conferencing applications, it is natural to structure programs in terms of multiple co-operating modules which communicate via inter-process communication techniques. In this sort of environment it is often convenient for management objects to request that transport services be instantiated for transport service access points (TSAPs) used by other parts of the application. In this case the new transport services must support a remote connect services (see Figure 4). For example, such a management entity may request that a remote display device's TSAP be connected to a another remote video server or camera.

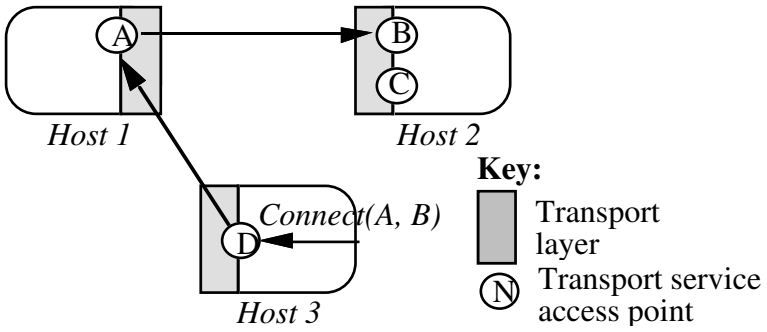

Figure 4: Remote connect scenario

Connection Establishment and Release

Connection establishment is a fully confirmed service similar in nature to that employed in the OSI reference model. Full option negotiation of QoS is performed on connection, and in keeping with the requirements for CM VCs, the negotiated QoS represents the required characteristics of the media in one direction only, from source to sink.

Primitives	Parameters
T-Connect.request	initiator-address, src-address, dest-address, protocol, class-of-service, QoS-tolerance-levels, vc-id
T-Connect.indication	"
T-Connect.response	"
T-Connect.confirm	"
T-Disconnect.request	initiator-address, vc-id
T-Disconnect.indication	initiator-address, vc-id, reason

Table 1: Connection establishment and release primitives and their associated parameters.

We now illustrate the implementation of the remote connect facility. Note that where it is required to form a VC in the conventional sense (i.e. where the initiator is also the sender), the caller simply sets the initiator to be the same as the source address.

When the initiating application generates a T-Connect.request primitive, this is relayed to the source entity, where on arrival a T-Connect.indication is issued to the application attached to the addressed TSAP. The source application can then accept the call with a T-Connect.response, or reject it with a T-Disconnect.request. If the request is accepted, the transport entity generates a T-Connect.request and from this point onwards the conventional connect protocol is followed. The outcome of this is eventually relayed back to the application which initiated the call. If the call is rejected then the initiating application receives a T-Disconnect.indication. A successful remote connect is illustrated in Figure 5. Note that the request may be rejected by the source, the destination or the network provider.

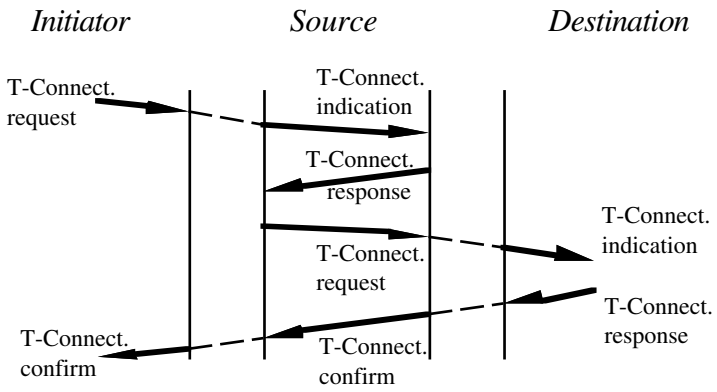

Figure 5: Successful remote connection establishment

It is also possible, of course, for an initiator to request that a VC be remotely released. A T-Disconnect.request is sent to the source or destination where on arrival a T-Disconnect.indication is issued to the attached application. The application may then issue a T-Disconnect.request which will be relayed to the other end-point, and on arrival will generate a T-Disconnect.indication.

Notification of QoS Degradation

If the class of service selected from the transport protocol for a particular VC incorporates the error indication facility, a primitive is required to convey to the transport user any errors or QoS degradations which may have occurred. This primitive is generated by the transport entity monitoring the VC over a suitable sample period. The primitive and its parameters are set out in Table 2. The primitive will identify the VC, provide the negotiated QoS tolerance levels, the sample period, the measured performance of the negotiated QoS tolerance levels within that sample period, and an error number to identify which of the tolerance levels have suffered degradation.

Primitive	Parameters
T-QoS.indication	initiator-address, src-address, dest-address, initial-QoS-tolerance-levels, sample-period, vc-id, current-QoS-tolerance-levels, error-number

Table 2: Primitive issued by transport service to notify transport user of degradation of negotiated QoS.

QoS Re-negotiation

For QoS re-negotiation, a fully confirmed service with full option negotiation similar to that used for connection establishment is employed. The primitive is detailed in Table 3. The time sequence diagram in figure 5 above is also applicable to the T-Renegotiate service.

Primitives	Parameters
T-Renegotiate.request	initiator-address, src-address, dest-address, new-QoS-tolerance-levels, vc-id
T-Renegotiate.indication	"
T-Renegotiate.response	"
T-Renegotiate.confirm	"

Table 3: Primitives employed for the re-negotiation of QoS. The primitives may be initiated by a transport protocol user, or by the transport protocol itself.

The transport protocol may need to tear down a VC and open a new one in order to supply the required QoS. Thus it should be capable of sustaining state so that it can re-synchronise once the new VC is established. If for some reason a modified service cannot be provided, the service request will be rejected with a T-Disconnect.request, and the transport user will receive a T-Disconnect.indication. However, in this latter case, the existing VC is *not* torn down; the T-Disconnect.indication simply indicates that the new service level requested can not be supported.

Note that the T-Renegotiate service allows the QoS of a VC to be modified but not its protocol type or class of service.

4.2 Orchestration services

This section presents an architecture which addresses the need for the co-ordination of multiple related CM transport streams and associated orchestration services. We distribute the functionality of the orchestrator over three architectural regions:-

- the distributed application platform,
- an area corresponding to the OSI upper layer architecture (ULA), and
- an area closely associated with the transport layer interface.

It can be seen from Figure 6 that orchestration is a multi-layered activity. Each layer provides *policy* to its lower neighbour and *mechanism* to its upper neighbour. This design provides both flexibility and efficiency because the lower layers are simply provided with targets, and all exceptions, error handling and re-structuring are handled in the layers above.

In essence, the operation of the three orchestration layers is as follows. The *high level orchestrator* (HLO) provides the view of orchestration seen by users of the applications platform. It is a location independent abstract data type (ADT) service interface which contains orchestration related named operations. Applications pass ADT interfaces, which represent connections in the object based domain, to these operations and the HLO arranges to have the required continuous synchronisation performed by the lower layers according to a *policy* specified by the application.

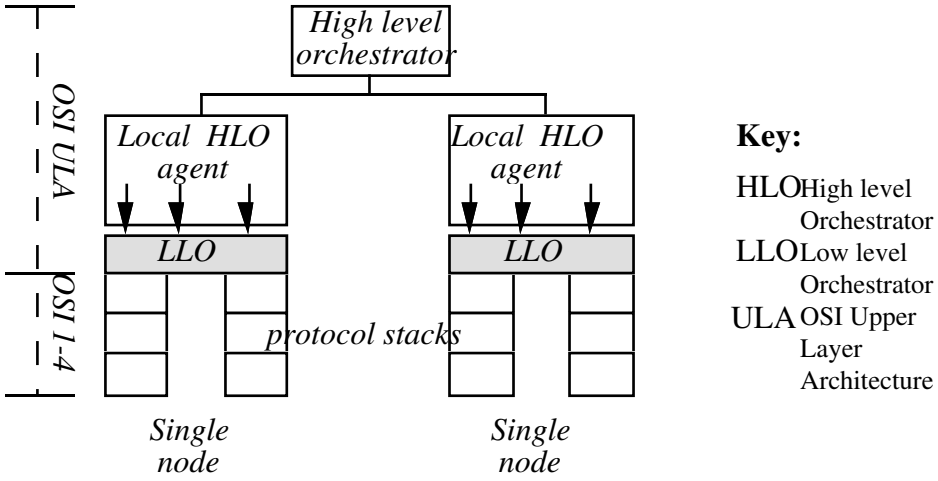

Figure 6: Three level orchestration architecture

Policies include constraints on how 'strict' the continuous synchronisation should be and actions to take on failure. The HLO is responsible for finding the physical locations of the connections underlying the given Stream interfaces, and thus choosing the node from which the lower levels of orchestration will be co-ordinated. The node selected, known as the *orchestration node*, is that common to the greatest number of VCs.†

† In our initial implementation we make a restriction that groups of orchestrated connections must have a common node, either at the source or the sink. With this restriction in force, we are able to use the clock at the common node as the datum for continuous synchronisation across connections, and use a simple clock synchronisation scheme. It should, however, be possible to lift this restriction without changing the basic architecture by including a general purpose clock synchronisation function (e.g. [Mills,89] within the orchestrator protocols. At the present time, however, it is not fully clear whether such a generalised facility is either required in practice or realistic in implementation.

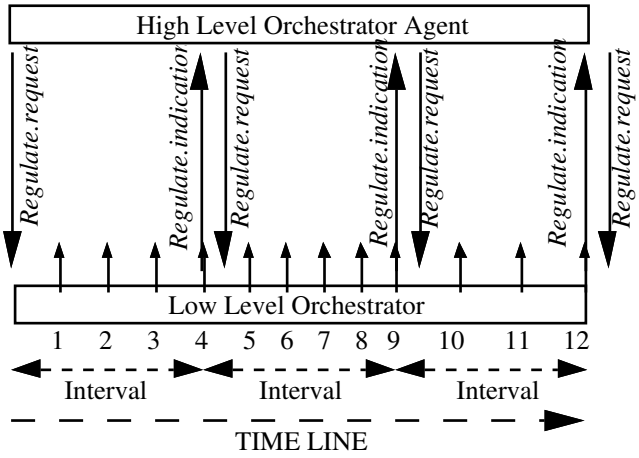

Figure 7: Interaction between HLO Agent and LLO.

Figure 7 illustrates the interaction between the HLO agent and the local LLO (note that Figure 7 only illustrates the interaction required for a *single* orchestrated connection; in general multiple connections will be involved). The HLO agent supplies the LLO with *rate targets* for each orchestrated VC over specified *intervals*. These targets ensure that each orchestrated VC runs at the required rate, relative to the master reference clock maintained at the orchestration node, for the required synchronisation relationship between the orchestrated VCs to be maintained. The LLO attempts to meet the required rate target over each interval for each VC, and reports back at the end of the interval on its actual success or failure. Then, on the basis of these reports, the HLO agent sets new targets for the next interval which compensate for any relative speed up or slow down among the orchestrated connections. The LLO operates on a *best effort* principle; it is the responsibility of the HLO agent to take appropriate action (e.g. set new targets or re-negotiate the connection QoS) if the LLO consistently fails to meet targets.The small numbered arrows in figure 7 represent the delivery of quanta of CM information which are released by the sink LLO instance to the application thread at times determined by the HLO initiated targets. These quanta are known as *chain-links*, and are the units of CM information meaningful to applications (e.g. video frame or text paragraph); the term is derived from a view of CM information as a *chain* of atomic *links* of information.

The low level orchestrator

The LLO orchestration interface consists of two sets of OSI-like primitives. It is envisaged that these primitives will correspond to an (as yet unspecified) area within the OSI ULA. In particular, the primitives are NOT intended as extensions to an OSI transport service. The two sets of orchestration primitives are as follows:-

- *primitives for priming, starting and stopping orchestrated groups of connections; and*
- *primitives for monitoring and regulating individual orchestrated connections.*

The first set operates over a grouping of transport connections. The primitives provide the ability to atomically *prime*, *start* and *stop* the flow of data in these connections both atomically and instantaneously. By 'priming' we mean that the receiver's buffers are initially filled with the required data for a particular connection, in preparation for a subsequent *start* command. This allows related media flows to be started simultaneously, which is a fundamental requirement for subsequent continuous synchronisation.

The second set of primitives operate on single transport connections in an orchestrated grouping. They enable the controlling HLO agent to set and monitor the above mentioned flow rate targets. As stated above, connections will attempt to meet these targets on a best effort basis. Primitives are also provided to report back to the HLO agent on the actual performance achieved at the end of each interval.

Orchestrate primitive

We assume that before an HLO agent attempts to instantiate an LLO orchestrating service, the VCs to be orchestrated have already been established. It is also assumed that the initiator of the LLO service (i.e. an HLO agent) resides on the common orchestration node. Table 4 summarizes the primitives involved and their parameters.

Primitives	Parameters
Orchestrate.request	orch-session-id, list-of-vc-ids
Orchestrate.confirm	"
Orchestrate.Release.request	orch-session-id
Orchestrate.Release.indication	orch-session-id, reason

Table 4: Orchestration request and release primitives together with their associated parameters.

The initiator issues a Orchestrate.request which causes an OPDU to be passed to the LLO instance at each source and sink of all VCs in the orchestrated group. This OPDU contains an orch-session-id supplied by the HLO. Each source and sink determines whether or not it is able to support the requested orchestration request, and replies to the initiating LLO. If accepted by all the remote LLO services, the HLO will receive a Orchestrate.confirm, or if rejected a Orchestrate.Release.indication giving a reason why orchestration was rejected. Rejection may occur because some LLO instance has no table space available, or because one or more of the specified VCs do not exist etc..

Orchestration is released by issuing a Orchestrate.Release.request. Orchestration will also be released implicitly if all the VCs in an orchestrated session are closed.

Group 1: Primitives for Priming, Starting and Stopping

The primitives to prime connections, and atomically start and stop the data flow on groups of orchestrated VCs are illustrated in Table 5.

Primitives	Parameters
Orch.Prime.request	orch-session-id
Orch.Prime.indication	"
Orch.Prime.response	"
Orch.Prime.confirm	"
Orch.Start.request	orch-session-id
Orch.Start.indication	"
Orch.Start.response	"
Orch.Start.confirm	"
Orch.Stop.request	orch-session-id
Orch.Stop.indication	"
Orch.Stop.response	"
Orch.Stop.confirm	"
Orch.Add.request	orch-session-id, vc-id
Orch.Add.indication	"
Orch.Add.response	"
Orch.Add.confirm	"
Orch.Remove.request	orch-session-id, vc-id
Orch.Remove.indication	"
Orch.Remove.response	"
Orch.Remove.confirm	"
Orch.Deny.request	orch-session-id, reason
Orch.Deny.indication	"

Table 5: Orchestration primitives for priming, starting and stopping.

Orch.Prime

The Orch.Prime primitive is used to ensuring that multiple streams of remotely stored CM data can be started together in a synchronised way. It is also useful in ensuring that time critical data can be pre-fetched and made available when required. A third application of Orch.Prime arises when it is required to clean out the buffers in an end to end connection. This need arises

when a user stops a media play-out and then wishes to seek to another part of the media before resuming. If the buffers were not flushed in this situation, a short burst of media buffered from the previous play would be discernible.

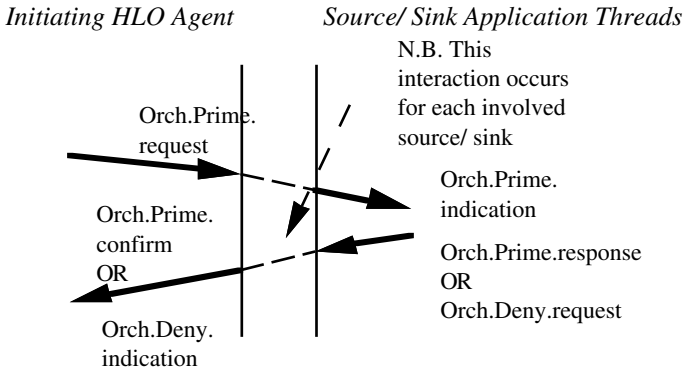

Figure 8: Orch.Prime.

The time sequence operation of the Orch.Prime primitive is shown in Figure 8. Orch.Prime.indication primitives are passed to the application threads associated with each end of each orchestrated VC. On receipt of the Orch.prime.indication, each application thread is expected to respectively start generating data for transmission or preparing to accept data. If any application thread is not in a position to do this it can reply with a Orch.Deny rather than a Orch.Prime.response. In either case, the result is passed back to the LLO at the orchestration node.

As data begins to arrive at the sinks, the sink LLOs allow the receiver's communications buffers to fill, but prevent the data from being delivered to the receiving application threads. At this point, the source will also be blocked by the protocol's flow control mechanism, but the pipeline is filled and ready to go.

Orch.Start

Having primed an orchestrated set of connections, the HLO will eventually issue a Orch.Start.request. This primitive re-starts the transport protocol and also unblocks the previously filled receive buffers so that they may be emptied by the sink application thread. As data is waiting at all the sinks, all the receiving application threads in the orchestrated group will start to receive data at (almost) the same instant.

Orch.Stop

Orch.Stop instantaneously freezes the flow of data in the specified VCs. Note, however, that the flow of data can not actually be stopped until the underlying protocol's flow control mechanism can take effect. As with the Orch.Prime primitive, the receive buffers are made unavailable to the application sink thread before they are drained so that data is available for a subsequent primed start. Also, the underlying protocol's flow control mechanism will eventually block the source application threads in the same way as Orch.Prime.

Orch.Add and Orch.Remove

The Orch.Add and the Orch.Remove primitives are employed to either add or remove a particular VC or VCs from an orchestrated group. The time sequence operation is again similar to figure 8. Note that when VCs are removed from an orchestrated group they are not disconnected and thus data may still be flowing.

Group 2: Primitives for Monitoring and Regulating

The primitives used to implement the regulation and monitoring functions on orchestrated VCs are illustrated in Table 6 and described in the following sub-sections. For brevity, we do not discuss the time sequence operation of these primitives.

Primitives	Parameters
Orch.Regulate.request	orch-session-id, vc-id, target-chain-link#, max-drop#, interval-length, interval-id
Orch.Regulate.indication	orch-session-id, vc-id, interval-id, chain-link#, dropped#, proto-block-times, app-block-times
Orch.Delayed.request	orch-session-id, vc-id, source-or-sink, interval-length, chain-links-behind
Orch.Delayed.indication	"
Orch.Delayed.response	"
Orch.Delayed.confirm	"
Orch.Event.request	orch-session-id, vc-id, event-pattern
Orch.Event.indication	"

Table 6: Orchestration Primitives for Regulation and Monitoring

Orch.Regulate.request

This primitive is issued by the HLO agent to set a flow rate target for the forthcoming interval for individual VCs in an orchestrated group. Parameters include the orchestration session ID, the ID of the VC to be controlled, the length of the forthcoming interval, and an interval-id to identify the corresponding Orch.Regulate.indication. The target-chain-link# parameter denotes the chain-link sequence number which should ideally be delivered to the sink application thread at precisely the end of the interval. This is the way in which the required flow rate target is specified: the required rate is calculated as ((*target-chain-link# - current-chain-link#*) / *interval-length*).

When the LLO is attempting to meet the requested flow rate target for a connection, there are three possible cases: it may be on target, behind target or ahead of target. If the connection is on target, no special action need be taken. If, however, either of the other cases is true, the following compensatory strategies are available to the LLO:-

- If a connection is behind, its sole compensatory strategy is to drop chain-links. The final parameter, max drop#, states the maximum number of chain-links which the VC may discard in order to achieve its flow rate target. All such discards are performed at the source by incrementing the source shared buffer pointer. This permits the source application thread to immediately insert another chain-link and thus overwrite the previous one before it is sent. This strategy may help a delayed VC to catch up and meet the link delivery target in the case that it is lack of transport bandwidth which is causing the delay.

- If, on the other hand, a connection is *ahead* of schedule, the compensatory action is simply to block. Note that, as with the Orch.Stop primitive, the flow control mechanism of the underlying transport protocol capable of rapid adaptation for this to be feasible. The rate based mechanism used in our protocol [Shepherd,91], has proved to be adequate for this purpose. Note also that, for both these compensatory actions, the LLO must take responsibility for attempting to spread compensatory actions over the length of the target interval to avoid unnecessary jitter.

If these compensatory actions are not available to the LLO (e.g. a max-drop# of zero will often be chosen where a no-loss medium such as voice is involved), then the necessary corrections must be taken by the HLO agent on the basis of the information gathered via the Orch.Regulate.indication primitive.

Orch.Regulate.indication

This primitive is used to report back to the HLO agent on the performance actually achieved by each orchestrated connection in comparison to the targets set by the HLO for the interval just completed. The interval-id parameter is used to match the indication to a prior request. The LLO will continuously generate Orch.Regulate.indications at the end of each interval length as established by the last Orch.Regulate.request. The statistics reported include the chain-link# *actually* delivered at the end of the interval, the number of chain-links *actually* dropped, and the

times spent blocking by both the application and protocol threads at both the source and sink ends of the VC.

The blocking time information is used by the HLO agent to determine which part of the system was responsible for any failure to meet the flow rate target. Based on this information, the HLO can take compensatory action if required.

Orch.Delayed

This primitive is issued by the HLO in response to the situation described above. The effect is to cause an indication to be delivered to the application thread(s) causing the delay. The intended interpretation of a Orch.Delayed.indication is that the thread is not running sufficiently fast to produce/ consume data at a rate required by the client of the location independent orchestration service.

Orch.Event

This primitive is used to register an interest in a particular application defined event associated with some chain-link; it thus provides support for event-driven synchronisation.

To register an interest in some application defined event, a Orch.Event.request is issued to an LLO instance at the sink end of an orchestrated VC, together with a value representing the event. Subsequently, the sink LLO instance will match this value, which is not interpreted in any way by the LLO, against the values in the event fields of the OPDUs associated with incoming chain-links on the set of orchestrated VCs. If the event in the chain-link matches the registered bit pattern, a Orch.Event.indication is raised.

5. Other considerations

Multimedia considerations stated in section 3.2 but not elaborated in the paper include group/multicast addressing, the high speed transport protocol, and some of the more traditional constituents of a complete transport system such as TSAP allocation, datagram services, data transfer mechanism and priority mechanisms, these are all for further study.

Conventional data transfer interfaces to transport services, such as the *send()* and *recv()* systems calls found in interfaces such as TLI [Olander,86] and Berkeley sockets [Berkeley,86], or the *data.request* and *data.indication* primitives in OSI, are not optimal for continuous media (CM) transmissions. This is because such interfaces have no model of an *on-going commitment* to service isochronous CM information. Each buffer of data to be transferred is treated as a separate entity with no particular temporal relationship to its successors and predecessors.

The conventional interfaces are also inefficient in a CM context. Each time a system call such as *send()* or *recv()* is issued, three pieces of information are passed [Govindan,91]: *synchronisation information* (i.e. it is implicit that the data transfer should commence 'now'), *data location* (i.e. the location of the data to be transferred), and *data transfer* (perhaps involving a copy to or from system space). However, it is possible in a CM systems to avoid specifying all this information on each send/ recv call. This is because the isochronous characteristics of application/ transport data transfers are implicitly known and thus the repetitive sequence of calls becomes unnecessary.

Our experiments in this area favour the adoption of a data transfer interface based around *shared circular buffers* with access contention between separate application and protocol threads controlled by semaphores.

The requirement for multicast arises from the nature of projected multimedia applications, many of which will be in the area of CSCW. In such applications a multicast facility is required for both transactional communication (RPC), and CM connections. In transactional multicasts, it may be required that any member of the multicast group should be capable of transferring information to all other members at any time. However, in a CM based multicast session a simple 1:N topology is usually all that is required. Appropriate support for group addressing must be provided in the

transport layer, but multicast support will be the responsibility of the underlying communications sub-system.

6. Summary

This paper outlines the design of a CM transport service and an associated orchestration service which permits real-time co-ordination between distinct transport connections. Our transport service has simplex VCs with flexible QoS configuration, including re-negotiation, and a facility for establishing connections remotely. Details of related work in the field of CM transport protocols may be found in [Wolfinger,91] and [Hehmann,91]. Our work is notable for the close integration of continuous media transport concerns with those of the distributed application platform that we consider an essential part of future systems building.

An orchestration architecture is described that consists of three components: a high level orchestrator which makes HLO services available from our object-based application platform, HLO agents which control and monitor orchestrated connections, and low level orchestrators which sample and regulate the flow of CM information over intervals as directed by the HLO agent. The orchestration system is architecturally separate from the transport sub-system although the two components must be intimately related in implementation.

The context of our work is that it forms part of the ESPRIT OSI 95 project. Part of the aim of this project is to develop new transport services suited to the new environment of high-speed multiservice networks and distributed multimedia applications. The ESPRIT OSI 95 project is not the only international attempt to introduce a new high-speed transport protocol: in particular, ANSI X3S3.3 has recently issued draft descriptions of a new high speed transport service (HSTS) and protocol (HSTP), which are to be studied further within OSI 95.

References

[Berkeley,86] Berkeley. "UNIX Programmer's Reference Manual, 4.3 Berkeley Software Distribution", Computer Systems Research Group, Computer Science Division, Univ. of California at Berkeley, CA, USA. April 1986.

[Blair,92] Blair, G.S., F. Garcia, D. Hutchison and W.D. Shepherd. "Towards New Transport Services to Support Distributed Multimedia Applications", *To be presented at Multimedia '92: 4th IEEE COMSOC International Workshop*, Monterey, USA, April 1-4, 1992.

[Hehmann,91] Hehmann, D.B., R.G. Herrtwich, W. Schulz, T. Schuett, and R. Steinmetz. "Implementing HeiTS: Architecture and Implementation Strategy of the Heidelberg High Speed Transport System" *Second International Workshop on Network and Operating System Support for Digital Audio and Video,* IBM ENC, Heidelberg, Germany, 1991.

[Mills,89] D.L. Mills, "Internet Time Synchronisation: the Network Time Protocol", *Internet Request for Comments No. 1129* RFC-1129, October 1989.

[Olander,86] Olander, D.J., G.J. McGrath, and R.K. Israel. "A Framework for Networking in System V." *Proceedings of the 1986 Summer USENIX Conference,* Atlanta, Georgia, USA, Pages 38-45.

[Shepherd,91] Shepherd, W.D., D. Hutchison, F. Garcia and G. Coulson. "Protocol Support for Distributed Multimedia Applications." *Second International Workshop on Network and Operating System Support for Digital Audio and Video,* IBM ENC, Heidelberg, Germany, Nov 18-19 1991.

[Wolfinger,91] Wolfinger, B., and M. Moran. "A Continuous Media Data Transport Service and Protocol for Real-time Communication in High Speed Networks." *Second International Workshop on Network and Operating System Support for Digital Audio and Video,* IBM ENC, Heidelberg, Germany, 1991.

[Williams,91] Williams, N., G.S. Blair. "Distributed Multimedia Application Survey", *Internal Report N. MPG-91-11.* Computing Department, Lancaster University, Bailrigg, Lancaster LA1 4YR, UK. April 1991.