Interoperability of Multimedia Information Systems

Frank Eliassen *
University of Tromso
Dept. of Computer Science
9037 Tromso
Norway
frank@cs.uit.no

John R. Nicol
GTE Laboratories
40 Sylvan Road
Waltham, MA 02254
USA
nicol@gte.com

Abstract

The goal of achieving openness is paramount in any large scale distributed information system featuring multi-vendor hardware and software. This is true for information systems interconnected over conventional distributed computing networks but also for systems interconnected over next generation broadband networks.

Unfortunately, in the case of general support for multimedia streams transport, the predominance of proprietary audio/video formats, compression schemes, networking protocols, and supporting hardware solutions, means that heterogeneity "reigns" for now. This situation has severe consequences on the interoperability of cooperating multimedia information systems.

In this paper we examine the issue of multimedia stream typing with a view to formalizing how to cope in a systematic way with the heterogeneity of isochronous information in broadband networks. Key areas we address are to device a means of rationalizing how it can be decided whether a given continuous media source and sink are computationally compatible, and whether their respective stated quality-of-service can be adjusted to find a mutually agreeable level.

1. Introduction

Future distributed multimedia applications supporting advanced interpersonal communication, will involve users at remote sites interacting across a wide area broadband network. It is highly unlikely that all these users will share a common infrastructure. For example, the supported audio/video formats, compression schemes, networking protocols, and supporting hardware solutions may vary considerably. This may even be the case within the same organizational boundary. For cooperating information systems, this situation may have severe repercussions for their ability to interoperate.

It will therefore be important to have in place a general connection management facility in support of distributed multimedia applications, capable of interconnecting endpoints exhibiting various forms of heterogeneity with respect to media stream properties and coding formats. We might refer to systems

* This work was undertaken while Frank Eliassen was a visiting professor at GTE labs from Oct 94 to Aug 95

including such support as *open*, since the range of properties of source and sink media flows it may connect could, in principle, be open ended. The facilities for overcoming end-point incompatibilities should therefore be part of the network. An important basis for providing this kind of support, is a proper type model for continuous media allowing stream end-point conformance and compatibility to be checked and resolved.

In this paper we outline the main principles of a type model for real-time continuous media flows which we believe must be embraced given the current absence of generally applicable conformance rules for stream interfaces. A salient feature of our approach is to interpret a type specification as a set of media descriptors. This allows us to define a variety of flow type relationships based on set theory. In particular we define subtyping rules supporting the definition of static conformance of stream interfaces, and a compatibility relationship that can be used as a basis for run-time flow property/QoS negotiations during binding. We also use the type system to define the properties of a facility supporting a dynamic coercer based type relationship. This allows an enhanced form of flow property/QoS negotiations that can be used to overcome forms of heterogeneity not supported by the above mentioned compatibility relationship.

Our approach to a type system for continuous media flows is guided by actual needs to solve problems of interoperability in distributed multi media applications. In relation to this, we illustrate the potential usefulness of the proposed type system and the defined type relationships in the context of a planned multi-party connection management facility supporting distributed multimedia applications in a heterogeneous environment. In such an environment participating applications may exhibit various forms of heterogeneity with respect to flow properties and coding formats.

The treatment of the type system will in this paper be kept at an informal level with a view to intuitive understanding rather than formal treatment. A detailed description of the type model is provided in a separate report [Eliassen95]

The remainder of the paper is organized as follows. In section 2 we discuss the requirements for typing of continuous media flows and review some related work. Section 3 presents the main ideas of a type model for real time continuous media flows, while in section 4 we define various flow type relationships based on this type model. A conformance relationship for stream interfaces is defined in section 5, and in section 6 we illustrate the use of the type system in a multi-way connection management facility. The last section concludes and discusses future work.

2. Typing of media flows

In distributed multi media applications, audio/video data must be communicated between interfaces producing and consuming flows in a timely manner. Until now, only operational interfaces have been supported in distributed application development platforms (such as CORBA [OMG91]). It is widely recognized, however, that operational interfaces are not suitable for conveying media streams. For example, contemporary implementation techniques of operational interfaces such as RPC, do not impose constraints on the timing of invocations. In order to uphold the semantics of a video source flow, an operational interface would require that invocations must be made at a certain rate. This will in general not be possible and so the notion of stream interface has therefore been proposed [Coulson92]. A stream interface imposes constraints itself on the timing of events, e.g. when the next audio sample should be produced or consumed. This is expressed as a QoS annotation specifying, for example, required sample rate. Furthermore, a stream interface is non-operational. It produces or consumes data elements *continuously* according to its QoS specification in cooperation with its environment (operating system and network) and some other (remote) stream interface(s). Stream interfaces have been adopted in the work on ODP [ODP95] and TINA-DPE [vanHalt95].

A stream interface consists of a collection of source and/or sink media flows. A continuous media flow (such as audio or video) is a time-based value interpreted as a (finite) sequence of temporally constrained data elements. The temporal dimension of the data elements refer to their required time and length of presentation. One can also refer to such a sequence as a *timed flow*.

For instance a stream interface modeling the behaviour of an audio input device such as a microphone, will consist of a single audio source flow, while a stream interface modeling the behaviour of a video phone, for example, will typically be modeled as a stream interface of one pair of video and audio source flows, and one pair of audio and video sink flows.

Stream channels are created between *compatible* stream interfaces for the purpose of exchanging audio and video (say) as dictated by the type and direction of the flows. The computational activity related to the creation of this channel, is normally referred to as *binding*. During binding, the interfaces to be bound must be typed checked for compatibility. Type checking here roughly means to ensure that the properties of each source flow are as expected by the corresponding sink flow. Flow properties include encoding information (i.e. data representation of the flow "on the wire") as well as information about resolution, presentation rates, etc. Type checking thus ensures that attempts to bind non-compatible interfaces do not succeed. For instance, it would be useless and wastful of resources to bind a source video flow that produces a particular variant of JPEG encoded images, to a sink flow that models an MPEG decoder. Type checking can also be used to resolve flow incompatibilities automatically by inserting one

or more appropriate *filters* into the flow. A filter can, for example, transform a flow from one format to the other.

Most of the work done until now on multimedia data modeling and presentation favors an object-oriented approach [Gibbs93]. However, such works typically applies the object-paradigm to facilitate implementation frameworks in which abstract classes and inheritance plays a major role. They offer little help by way of resolving issues of interoperability of multimedia systems since the interpretation of the content of audio and video objects (say) usually is based on attribute values of the user-defined multimedia object itself as opposed to being supported through a type system of the data model. This makes it difficult (if not impossible) to define general conformance rules between stream interfaces in distributed multimedia systems.

Within the standards arena, the need to declare the properties of interfaces for conformance checking as part of a system evolution process and the binding of interfaces, has already been recognized for example in the Reference Model of Open Distributed Processing [ODP95]. Unfortunately, complete interface conformance rules have been defined for operational interfaces only. Conformance rules for stream interfaces have been deemed outside the scope of the standard. However, the advantages of detecting type errors at compile (or bind) time rather than at run-time, is no less for stream interfaces than for operational interfaces. Other works that recognize the need for type checking of continuous media flows during binding, include [Bates95] and [Coulson92]. However, they offer no generally applicable definition of the notion of compatibility.

Nevertheless, standardization obviously has an important role to play in the solution of interoperability problems. The "right" standard type system for continuous media flows would constitute a framework within which new standard flow formats can be developed, and it would offer universal conformance and compatibility rules also applicable to these forthcoming standard formats.

3. A type model for multimedia flows

We take a flow to be a (finite) sequence of time-based data units called *elements*. Each element is of a particular generic type such as an audio sample or an image frame. A flow may be heterogeneous, i.e. it may contain elements of various types. For example MPEG can be modeled as a flow in which several different types of image and audio elements are multiplexed. Associated with each generic element type is a set of "standard" attributes and value constraints (the *element descriptor*). The attributes describe properties of the element type such as encoding, resolution and depth for image frames, and sample-size and number of channels for audio samples. The value constraints specify how the value of

some attributes constrain the value of other attributes. For example, certain raster image encodings allow only specific image depths.

We also associate a set of standard attributes and value constraints with the flow as a whole (the *flow descriptor*). Attributes of the flow descriptor include image rate and duration. The flow descriptor also includes a specification of sequencing constraints on the elements in the flow. For example an MPEG flow must be divided into groups of frames such that each group starts with a specific kind of frame referred to as an I-frame. Sequencing constraints are specified using *sequencing expressions*, similar to regular expressions.

A "missing" attribute in a media descriptor of a flow type specification, is interpreted as "don't care", i.e. any value from the domain of the missing attribute is supported. Similarly, missing sequencing constraints in a flow specification means that any sequence of the declared element types of the flow, is legal.

In summary of the above description, the following example is a partial specification of an MPEG flow. For simplicity, we have omitted value constraints and most of the element and flow attributes.

```
type MPEG = Flow[I:Image[encoding:mpeg_i]
                + P:Image[encoding:mpeg_p]+B:Image[encoding:mpeg_b]]
                      [image_rate: {24,25,30,50}]
                                        (I[1]&(P[+]&B[*] | P[*]))[*]
```

The above specification states that an MPEG flow consists of three different image frame types labeled I, P, and B respectively. The element descriptors specify encodings, while the flow descriptor specifies image rates and sequencing constraints. The latter is specified as a sequencing expression indicating that an MPEG flow consists of zero or more sub-sequences (the outer [*] operator), such that each group of frames must start with one I-frame, followed by either one or more P-frames followed by zero or more B-frames, or by zero or more P-frames..

Each image type in the example has a different encoding. In general, we assume that the value on the encoding attribute is a globally unique name determining the structure and semantics of the content of the element (including element headers). In support of such a scheme one would probably require a global naming authority in which element formats (consisting of element header and element data) can be registered. The names themselves could be ASN.1 object identifiers, for example. As a side bar, the JPEG standard alone will give rise to about 29 different image frame formats, given its many compression and encoding options (see for example [Brown95, p.383]).

In general, an attribute value can be specified as a set of "atomic" values. This reflects our desire to let a flow type specification represent a set of possible flow properties. For example when a sink flow type specifies a set of different names

on the image encoding attribute, it actually declares that it can accept flows where the images can have any of the indicated formats. In the above example, this feature is illustrated by the `image_rate` attribute. The elements of the set represent alternative acceptable image rates. We may therefore *interpret* a flow type as a set of optional flow properties.

e = [a : {1,2,3}, b : {A,B}] { a.<{1},{2}> => b.<{A}, {B}> }

I(e)

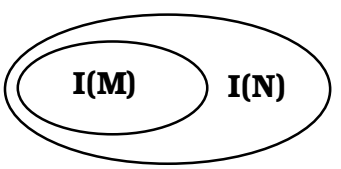
a	b
1	A
2	B
3	A
3	B

Figure 1: Interpretation of media descriptor

The interpretation of a flow type is based on the interpretation of the media descriptors occurring in the flow type specification. Figure 1 shows an abstract media descriptor *e* with value constraints, and its associated interpretation *I(e)*. The value constraint specifies how the *b* attribute is constrained by the *a* attribute. The table *I(e)* may be automatically derived from a specification as a set of tuples (similar to a table in a relational database). Details of this can be found in [Eliassen95]. Each tuple represent a possible combination of attribute values satisfying the specification. We take the interpretation of a flow type to be the cartesian product of the interpretations of its media descriptors where tuples not satisfying the flow constraints, are filtered out[1]. We interpret sequencing expressions much the same way we interpret media descriptors, i.e. as the *set* of possible element sequences of a flow.

4. Flow type relationships

Interpreting a flow type as a set of flow properties, allows us to define flow type relationships based on set theory. For example, the usual semantics of the subtype relationship is that of set inclusion. Applied to flow types we may therefore define a flow type M to be a subtype of the flow type N, if the interpretation of M, I(M), is a subset of the interpretation of N, I(N). This is illustrated in figure 2 using a Venn diagram.



[1]This is a conceptually "simpler" definition than the corresponding definition in [Eliassen95] in the sense that the latter is closer to a possible implementation scheme than the former. In [Eliassen95], media descriptor interpretations are compared directly without being first combined, to detect type relationships.

Figure 2: M is a subtype of N when I(M) is a subset of I(N).

Since missing attributes in a flow type specification is interpreted as "any value" (c.f. above), the subtype must specify at least the attributes of the supertype. Also it is sufficient to compare their interpretations projected to their common attributes.

An example of flow types (without value constraints) that are subtype related, are the following:

```
Flow [Image [width:{640,320},height:480] ]
                [segmentsize:(3..5), rate:4]
   < Flow [Image [width:{155,320,640}] ] [segmentsize:(1..5)]
```

We may also define a relaxed flow subtype relationship where the subtype may support less frame types than the supertype. This relationship is meaningful under the assumption that a sink accepts flows where some frame types can be totally missing as compared to its signature. In the example below, the flow type `MPEG_V` is a relaxed subtype of the flow type `MPEG_AV`. In this case, observe that the interpretation of the sequencing constraint of `MPEG_V` is a subset of that of `MPEG_AV`.

```
type MPEG_AV = Flow[I:Image[encoding:mpeg_i]
                    +P:Image[encoding:mpeg_p]
                    +B:Image[encoding:mpeg_b]
                    +Audio[...] ] [image_rate: 25]
                          (I[1]&P[+]&B[*]&A[*])[*]
type MPEG_V = Flow[I:Image[encoding:mpeg_i]
                    +P:Image[encoding:mpeg_p]
                    +B:Image[encoding:mpeg_b]][image_rate:{25,30}]
                           (I[1]&(P[1]&B[2])[*]
```

A flow subtype relationship as indicated above, is useful for checking the consistency and correctness of evolving distributed multi media systems. For instance when upgrading a multimedia service, the new version of the service must conform (in the substitutable sence) to the old version of the service to guarantee the continued servicing of existing clients.

The binding of source and sink flows, however, requires greater flexibility than is provided by static subtyping rules. For instance, if a source flow and sink flow supports a number of different flow properties, it would generally be too restrictive to require that the source supported set of flow properties must be a subset of the sink supported set of flow properties to allow the binding between them to take place. A more appropriate approach would be to allow attempts to bind if the source and sink support at least one common kind of flow (note that the attempt to bind may subsequently fail due to lack of resources).

To satisfy this requirement we define a compatibility relationship between flow types. We define a flow type M to be compatible with a flow type N if the interpretations of M and N have non-empty set intersections. This means that M and N share at least one common flow property (see figure 3).

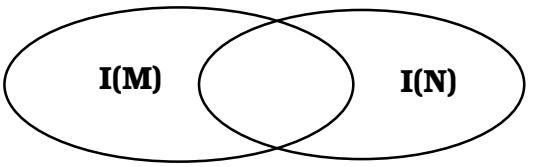


Figure 3: M and N are compatible when the intersection of I(M) and I(N) is not empty.

Examples of flow types that are compatible, are the following:

```
Flow [Image [width:{640,320},height:480]][segmsize:(3..5),rate:4]
<>
Flow [Image [width:{155,320}]][segsize:(1..4)]
```

The relaxed flow compatibility relationship allows element types to be dropped in either of the two flows to achieve "strict" compatibility. Mandatory element types (as specified in a sequencing constraint) may, however, not be dropped from a flow. For example, I-frames may not be dropped from an MPEG flow. Two relaxed compatible flow types are the following:

```
type MPEG_AV = Flow[I:Image[encoding:mpeg_i]
                    +P:Image[encoding:mpeg_p]+A:Audio[...]]
               [image_rate:{25,30}] (I[1]&P[2]&A[*])[*]

type MPEG_V = Flow[I:Image[encoding:mpeg_i]
              + P:Image[encoding:mpeg_p]+B:Image[encoding:mpeg_b]]
              [image_rate: {20,25}] (I[+]&(P[2]|P[3])&B[*])[*]
```

Whether to allow relaxed flow type relationships or not as a basis for binding and conformance, we consider being a matter of "specific system policy". In [Eliassen95] we show how sequencing constraints can be used to declare such a policy.

Binding based on compatibility has the limitation that the source and the sink flow must support a common flow property to allow the binding to take place. This may be a severe hurdle for providing video services, for example, to an open community of clients whose capabilities with respect to supported flow properties, may be unknown in advance. Over the years, however, a considerable number of filters transforming between different media formats, have been developed. The purpose of these filters is to overcome differences in formats of received (multi-media) documents, and formats handled by the presentation hardware and software in the receiving system. As such, filters provide an important means by which to overcome important aspects of heterogeneity.

The use of format transforming filters in current applications is typically hard-coded into the application code, leaving little or no room for flexibility and easily provided extensibility. We therefore believe that the application of filters should be supported at the system level as a generic concept allowing new filters to be easily integrated and exploited. This should not be left to the application programmer alone to handle.

A filter based type relationship may be exploited during flow property negotiations in a way that cannot be supported within the framework of any of the type relationships mentioned above. That is, if a source flow and a sink flow to be bound are incompatible, an extended form of system support would be to perform automatic determination of whether there exists some filter(s) that could be inserted into the flow. For example, source and sink image flows with no common image encodings could still be bound in those cases where an appropriate image encoding filter can be provided as a real time interceptor. This is illustrated in figure 4 in terms of flow type interpretations.

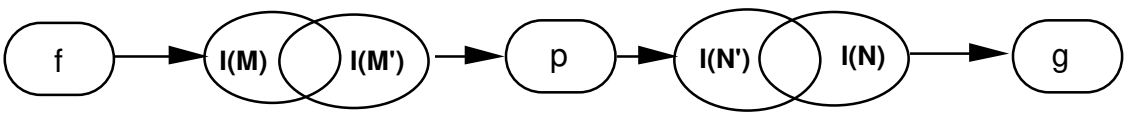


Figure 4: Indirect binding via filter of incompatible flow end-points

The type M of the source flow *f* is assumed to be incompatible with type N of the sink flow *g*. If a filter *p* with sink flow type M′ compatible with M, and source flow type N′ compatible with N, can be inserted into a flow between *f* and *g* as real time interceptor, the two end-points can be bound indirectly via the filter *p*.

The application of filter-based flow type relationships to binding, requires the support from a dynamic knowlegdebase to help determine the existence of the required filter(s). The knowledgebase will, in general, be graph structured (collection of directed graphs) where the nodes represent frame and flow types and the edges represent filters. Suppose *S* and *T* are two nodes in the knowledge base. The interpretation of an edge from *S* to *T* labeled *f* is that *f* is a filter that takes arguments of flow type *S* and produces an output of flow type *T*.

5. Conformance checking

Conformance rules express conditions for substitutability. Thus a service T conforms to a service S if S can be replaced by T such that existing clients depending on the services of S may still be serviced by T without modification to their code.

It follows that an important requirement to the relationship between conformance and compatibility is that when replacing a multi media service S with a conforming service *T* the compatibility of flow functions must be preserved form *S* to *T*. This means that clients that used to bind to flow end-points of the *S* service must be able to continue to do so for flow end-points of the *T* service, and without modifications to their supported flow properties. The only way we can guarantee this, is to require that the new service *T* must provide at the least the flows provided by the old service *S*. Furthermore, for each flow *f* provided by *S*, the type of the corresponding flow *g* of *T* must be a supertype of the type of *f*.

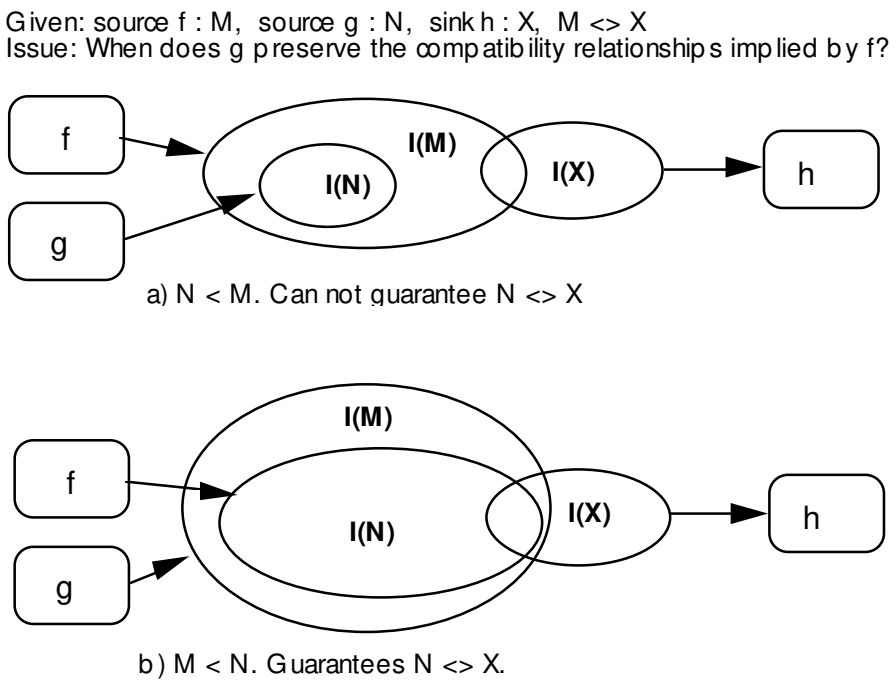


Figure 5: Conformance rules for flows

This is illustrated in figure 5 for a source flow *f* to be replaced by a source flow *g*. The scenario illustrated by the figure also assumes the existence of a client having a sink flow *h* such that *f* and *h* are compatible. This represents the fact that *f* and *h* can be directly bound. The problem is now if we want to replace the service represented by the source flow *f* by some other source flow *g*, what is the required type relationship between the flow types of *f* and *g* to guarantee the continued service of the client with the *h* flow. Figure 5a illustrates the case where the flow type N of g is a subtype of the flow type M of *f*, while figure 5b illustrates the case where M is a subtype of N. We see that only the latter case can guarantee continued compatibility with existing clients. The case where *f* and *g* are sinks and *h* a source, can easily be checked the same way as above. The case where *f* and *g* are filters applies as well as a combination of the two cases above.

6. Application of the type model

In this section we illustrate potential usage of the type model introduced above in the context of a multi-way connection management facility that are currently being developed at GTE Laboratories [Gutf95]. We assume it is required that the connection facility supports audio/video conferencing in an environment where participating applications may exhibit various forms of heterogeneity with respect to flow properties and coding formats.

Based on the architecture in figure 6, we give a scenario of the interactions that take place when the application P1 (say) joins an pre-existing conference. The

multi-party stream binding object is an abstraction of the stream service provider (i.e. the broadband network).

When P1 wants to join a conference, it invokes the "join conference" operation of its local agent (interaction 1). The local agent retrieves the interpretation of the flow properties supported by P1, from it local MIB (interaction 2). The local agent then invokes the "join conference" operation of the connection manager on behalf of P1 (interaction 3) supplying the retrieved flow properties from the local MIB as arguments.

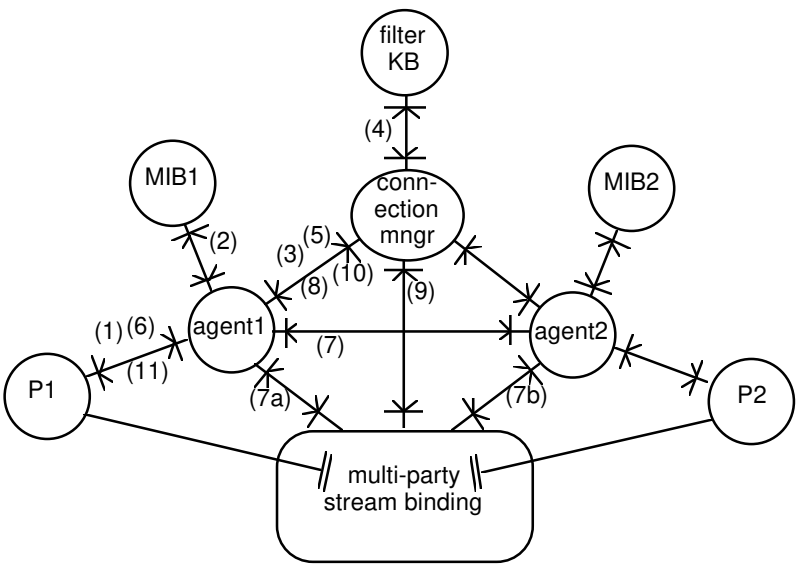


Figure 6: Architecture of conference facility

The connection manager compares the received flow properties with the supported flow properties of each of the participants in the conference to decide compatibility. In cases where there is incompatibility, the filter knowledge base is consulted to search for appropriate filters that can resolve the incompatibilities (interaction 4).

The result of the request is returned to the agent (interaction 5). A negative result (due to unresolvable flow incompatibilities, say) is immediately returned to P1 (interaction 6). In that case the "join conference" operation fails. A positive result, however, indicates that the flow properties supported by P1 are compatible (directly or via filters) with those of the current participants. In that case the negotiable set of flow properties together with information about needed filters (if any) is returned as part of the result (interaction 5). This information may be saved by the agent for possible later re-negotiation of QoS during the conference. Based on the returned information from the connection manager, the agent now starts flow property negotiations with each of the local agents of the current participants (interaction 7). In cases where indirect binding via filters must be used, the negotiations are indirect via the multi-party stream

binding object (interactions 7a and 7b) since we assume it is the responsibility of that object to provide the processing resources to execute the filters.

If the negotiation fails, this is reported back to the application as a failed "join conference" attempt. If, on the other hand, the negotiations succeed, the agent starts negotiations for network resources satisfying the negotiated application level flow properties. Details of this is not shown here. If the negotiation for network resources also succeeds, the agent requests the conference manager to establish the corresponding connections (interaction 8). The conference manager interacts with the multi-party stream binding object to make the connections and to activate the negotiated filters at special network nodes (interaction 9). At this point P1 is registered as a new participant of the conference. Finally the connection manager reports back to the agent (interaction 10), which again reports back to P1 (interaction 11).

An important characteristic of the above scenario is that the network resources are negotiated only after the application level flow properties have been agreed upon. This prevents from holding expensive and shared operating system and network resources until it is clear that the negotiation responder (i.e. agent2) have appropriate I/O devices, processing capacity and storage space available to support the flow properties proposed by the negotiation initiator (i.e. agent1). Only after the negotiation have completed, are corresponding stream channels and flows created.

7. Conclusions and future work

In this paper we have proposed the main principles of a type system for real-time continuous media flows. It is our belief that such a type system constitutes an important part of the basis needed to meet the goal of universal interoperability of multimedia information systems. This includes the ability to express the precise meaning of a flow type and flow subtype, flow type compatibility, and flow property negotiations. We illustrated the potential usefulness of the type system in the context of a multi-way conference facility.

A salient feature of our approach is to interpret a type specification as a set of media descriptors. This allowed us to define a variety of flow type relationships based on set theory. In this paper we have defined flow subtyping rules supporting the definition of static conformance rules for stream interfaces, and a weaker compatibility relationship, suitable as a basis for run-time flow property negotiations. An important observation we made is that a subtype relationship for continuous media flows seems too restrictive as a basis for flow property negotiations during stream interface binding. The flow type relationship called compatibility was therefore introduced.

We also used the type system to define the properties of a facility supporting a dynamic coercer based type relationship. This allowed an enhanced form of flow

property negotiations that can be used to overcome forms of heterogeneity not supported by the above mentioned compatibility relationship.

Judging from our current analysis, application of the type model and associated type relationships in the conference facility, will be beneficial. The multi-way connection management facility will operate in an environment where participating applications may exhibit various forms of heterogeneity with respect to flow properties and coding formats. In such an environment, the type system can be used to decide whether flow capabilities of a new participant are compatible with those already in the conference. This checking can be done at the time a new participant requests to join a conference, and before involving operating system and network resource negotiations.

The main motivation for the type system described in this paper was the desire the create a formal basis for achieving universal interoperability of multi media applications. Obviously standardization is crucial for the solution to this problem. In our future we will work address the issue of "standardizing" the possible generic element types of a flow and the corresponding element and flow descriptors, as well as a common transfer format for flows that can be universally understood. It should be observed, however, that the type model we propose does not dictate a particular solution to these issues.

In our future work we will also concentrate on practical applications of the proposed type system. This includes development of concrete policies and algorithms for flow property negotiation and optimal coercer selection for resolving flow incompatibilities. We also envisage the type system will be useful for flow property based trading in which a client may either trade for conforming stream interfaces for the purpose of locating a continuous media server that can substitute some other server in a given environment (say), or for compatible stream interfaces to which the particular client may bind.

References

[Bates95] Bates, J., Bacon J., *Supporting Interactive Presentations for Distributed Multimedia Applications,* Multimedia Tools and Applications, Kluwer, **1**(1), March 1995, pp. 47-78.

[Brown95] Brown, C.W., Shepard, B.J., *Graphics File Formats,* Prentice Hall, 1995.

[Coulson92] Coulson, G., Blair G. S., Stefani, J. B., Horn, F., Hazard, L., *Supporting the Real-Time Requirements of Continuous Media in Open Distributed Processing*, Technical Report MPG-92-35, University of Lancaster, 1992.

[Eliassen95] Eliassen, F., Nicol, J.R., *Polymorphic Typing for Continuous Media Flows and its Application to QoS Brokerage,* GTE Laboratories Technical Report TR-0303-07-95-380, July 1995.

[Gibbs93] Gibbs, S., Breitender, C., Tsichritzis, D., *Audio/Video Databases: An Object-Oriented Approach*, Proc. 9th IEEE Int'l Conf. on Data Engineering, 1993, pp. 381-390.

[Gutf95] Gutfreund, S., Nicol, J.R., Phuah, V., Eliassen, F., *ATOMS: ATM Support Mechanisms for Advanced Multimedia Applications Research,* GTE Laboratories Technical Report TR-0297-06-95-380, June 1995.

[ODP95] ISO/IEC JTC1/SC21, *Draft Recommendation X.903: Basic Reference Model of Open Distributed Processing - Part 3: Architecture.*

[OMG91] Object Management Group and X/Open, *The Common Object Request Broker: Architecture and Specification*, OMG Document Number 91.12.1 (Revision 1.1), December 1991.

[vanHalt95] van Halteren, A, Leydekkers, P., Korte, H., *Specification and Realisation of Stream interfaces for the TINA-DPE* In proceedings of TINA'95, Melbourne, Australia, Feb. 1995, pp. 299-314.