Supporting Openness in Distributed Multimedia 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. However, 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 for the interoperability of distributed multimedia systems.

In open distributed systems, the notion of interface type is important for supporting conformance checking of interfaces during compile (or bind) time. Unfortunately, generally applicable conformance rules for interfaces including continuous media flows, have received relatively little attention in the multimedia research community.

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 distributed systems. A key aspect of the work is to devise a type model that can be used to decide whether a given continuous media source and sink are computationally compatible. We also consider architectural support needed to interconnect flow end-points that exhibit various forms of heterogeneity with respect to media stream properties and coding formats.

Keywords: Openness, Multimedia, Type model, Type relationships, Architectural support.

1. Introduction

In an open distributed system, there is no guarantee that components are built using the same technology. This is true for distributed systems interconnected over conventional computing networks, but also for systems interconnected over the next-generation broadband networks. In particular, future distributed multimedia applications supporting advanced interpersonal communications, will involve users at remote sites interacting across a wide area broadband network. Concerning such applications, the supported audio/video formats, compression schemes, networking protocols, and supporting hardware solutions may vary considerably—a situation that will have severe repercussions for different systems' ability to interoperate.

It would therefore be ideal to have in place a general connection-management facility in support of distributed multimedia applications, capable of interconnecting endpoints that exhibit various forms of heterogeneity with respect to media stream properties and coding formats. We might refer to systems 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. An important foundation on which to build this kind of support, is a proper type model for continuous media—a model allowing stream end-point conformance and compatibility to be checked and resolved.

[*] This work was undertaken while Frank Eliassen was a visiting professor at GTE Laboratories Incorporated from October 94 to August 95.

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 supports the definition of 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 model 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 aforementioned compatibility relationship.

We illustrate the potential application of the proposed type model and the defined type relationships in the context of an architecture supporting continuous media connections in a heterogeneous environment. In such an environment flow end-points may exhibit various forms of heterogeneity with respect to flow properties and coding formats.

Treatment of the type model will be kept informal in this paper with a view to presenting an intuitive understanding of our approach. A more detailed and formal description of the type model is presented 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 given architecture. The last section presents our conclusions and discusses future work.

2. Typing of media flows

A common requirement of distributed multimedia applications is that audio/video data must be communicated between component application interfaces producing and consuming data streams 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 in nature: 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 [vanHalteren95].

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 an audio input device such as a microphone, will consist of a single audio source flow, while a stream interface modeling 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 (for example) 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 type-checked for compatibility. Type-checking here informally refers to the process of ensuring 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 usually be useless and wasteful 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.

Although the benefits of detecting type errors at compile (or bind) time rather than at run-time are widely recognized, generally applicable conformance and compatibility rules for stream interfaces have received relatively little attention in the multimedia research community. 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 is usually 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. To each generic element type is associated 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 elements, and sample-size and number of channels for audio elements. 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.

There is also a set of standard attributes and value constraints associated to the flow as a whole (the *flow descriptor*). Attributes of the flow descriptor include image rate and duration. The flow descriptor also include 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 element 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 each group of image elements must start with a 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, identifying registered format specifications in an special purpose database. As a side bar, the JPEG standard alone will give raise to about 29 different image element 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.

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

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)*. This table (similar to a relation) may be automatically derived from a specification. Details of this can be found in [Eliassen95]. Each tuple represents a possible combination of attribute values satisfying the specification. The value constraint specifies how the *b* attribute is constrained by the *a* attribute. We take the interpretation of a flow type to be the cartesian product of the interpretations of its media descriptors[1]. This interpretation of a flow type allows us to define a variety of flow type relationships such as subtyping and compatibility, based on set theory. We pursue this notion in the next section.

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.

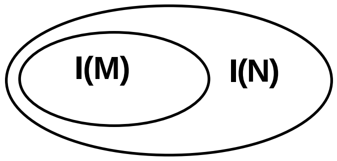


Figure 2: M is a subtype of N when I(M) is a subset of I(N).

Since missing attributes in a flow type specification are interpreted as "any value" (as explained earlier), 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 element types than the supertype. This relationship is meaningful under the assumption that a sink accepts flows where some element 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.

```
type MPEG_AV=Flow[Image[encoding:mpeg_i]+Image[encoding:mpeg_p]+
                  Image[encoding:mpeg_b]+Audio[...] ]

type MPEG_V=Flow[Image[encoding:mpeg_i]+Image[encoding:mpeg_p]+
                  Image[encoding:mpeg_b] ]
```

A flow subtype relationship as indicated above, is useful as a basis for checking the consistency and correctness of evolving distributed multimedia systems. For instance when upgrading a multimedia service, the new version of the service must conform (in

[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.

the *substitutable* sense) to the old version of the service to guarantee the continued servicing of existing clients (for details see section 5).

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 attempt 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.

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

```
Flow [Image [width:{640,320},height:480] ]
                        [ segmentsize:(3..5), rate:4]
<> Flow [Image [width:{155,320}] ] [segmentsize:(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_AV1= Flow [Image [encoding:mpeg_i]+
                        Image[encoding:mpeg_p]+Audio[...] ]

type MPEG_V2 = Flow [Image [encoding:mpeg_i]
                  +Image[encoding:mpeg_p]+Image[encoding:mpeg_b] ]
```

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, has been developed. The purpose of these filters is to overcome differences in formats of received (multimedia) 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 3 in terms of flow type interpretations.

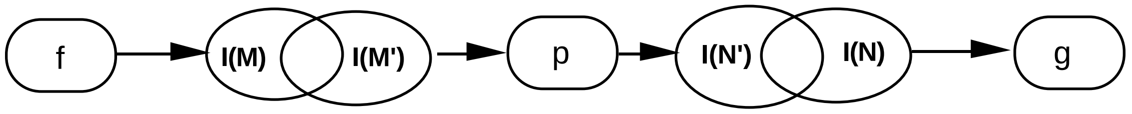


Figure 3: Indirect binding of incompatible flow end-points via filter

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 knowledge-base to help determine the existence of the required filter(s). The knowledge-base will, in general, be graph structured (collection of directed graphs) where the nodes represent element and flow types and the edges represent filters. For example, if *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*. In an open distributed system, one may refer to such a knowledge base as a *trader*. A trader locates service providers in a distributed environment satisfying a given set of requirements such as their type (see also section 6).

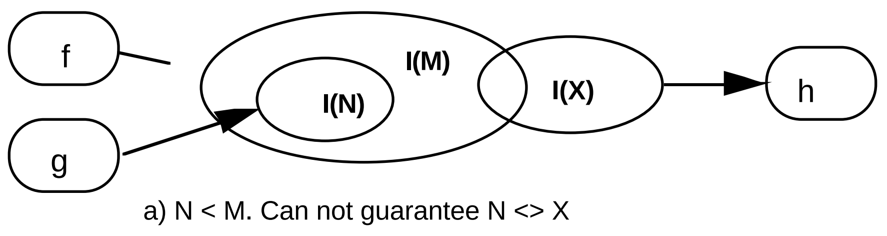
5. Conformance checking

Conformance rules express conditions for substitutability. This means that a service *S* may be transparently replaced by another service *T* as long as clients depending on the services of *S* may still be serviced by *T* without modification to their code. In such cases *T* is said to conform to *S*.

It follows that an important requirement to the relationship between conformance and compatibility is that, when replacing a multimedia 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.*

This is illustrated in figure 4 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. If we wish to replace the service represented by the source flow *f* by some other source flow *g*, the problem is now to determine 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 4a illustrates the case where the flow type N of *g* is a subtype of the flow type M of *f*, while figure 4b 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.

Given: source f : M, source g : N, sink h : X, M <> X
Issue: When does g preserve the compatibility relationships implied by f?



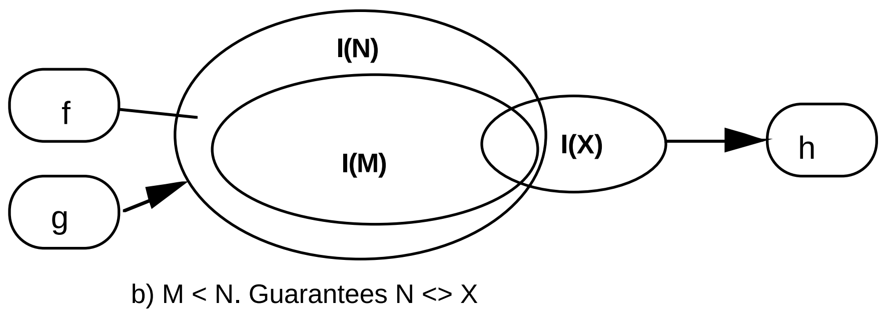


Figure 4: Conformance rules for flows

6. Architectural support for typed flows

In this section we consider architectural support needed to (transparently) interconnect flow end-points that exhibit various forms of heterogeneity with respect to media stream properties and coding formats. As indicated above, the basic architectural components needed is a multimedia trader that may locate continuous media services based on requirements of flow type conformance, and network services that may act as real time filters in a flow to transform from one flow format to an other. Such architectural components may be integrated into connection-management architectures in different ways depending on the nature of the connections provided. For example, the way these components are integrated into a multi-party connection-management scheme need not be suitable for a point-to-point connection-management scheme.

In this paper we assume a simple point-to-point connection-management scheme. Furthermore, rather than attempting a general discussion of possible ways to integrate multimedia trading and flow filtering into a point-to-point connection-management scheme, we give a hypothetical scenario of the interactions that take place when a stream binding is established between to incompatible stream interfaces. The scenario is based on the architecture shown in figure 5. We assume that the application C (the customer) wants to use a Video-On-Demand (VOD) service featuring a particular movie. The problem for C is that the properties of the source flow of the VOD is incompatible with the properties of its own sink flow. As indicated earlier in this paper, the problem can be solved if one or more appropriate filters can be inserted into the flow between C and the VOD server. The filter must transform from the format supported by the VOD server to the format supported by C. We assume in the following that the VOD service takes the initiative to solve this problem.

To request the playback of the movie, C interacts with the VOD service over some operational interface. Part of the information provided by C is a specification of its supported sink flow properties. The VOD service compares the received sink flow properties with its own supported source flow properties. If there is compatibility, C and the VOD service may directly negotiate the flow properties to be used over a direct stream binding between them (this case is not shown in the figure). The negotiable flow properties are those they have in common. In the case where there is incompatibility, the VOD server consults the multimedia trader to search for an appropriate filter service that can resolve the incompatibilities.

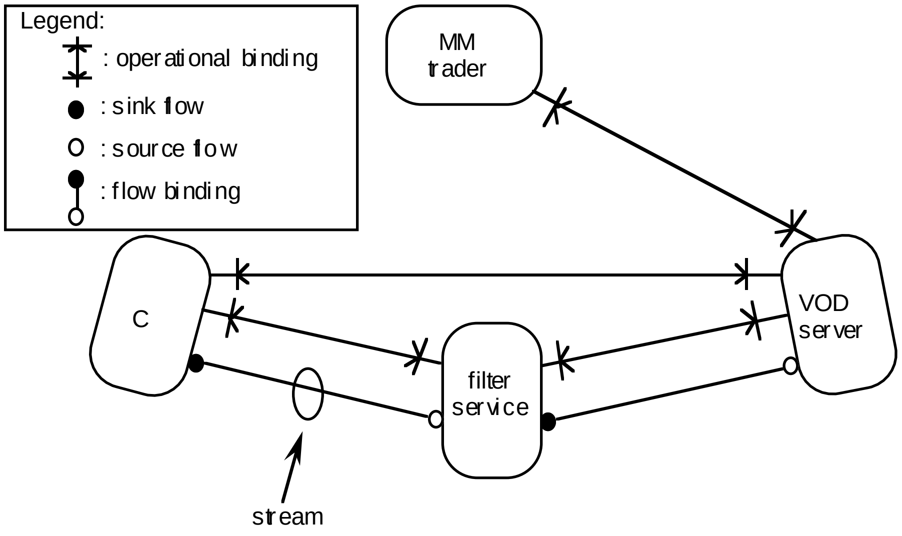


Figure 5: Trading and filtering in typed stream binding

The appropriate filter service must support a sink flow type compatible with the source flow type of the VOD server, and a source flow type compatible with the sink flow type of C. If the multimedia trader is able to locate such a service, it returns its identity to the VOD service. Otherwise the VOD server will not be able to service the request from C.

The VOD server must now request the filter service to perform the necessary transformations of the flow between C and itself. If the filter service is available (i.e. has sufficient resources), the VOD server will inform C that the requested service will be provided via the particular filter service. The filter service on the other hand, must negotiate with both C and the VOD service to agree on the flow properties to be used over the two "chained" stream channels. If the negotiations succeed, the stream channels can be established.

An important characteristic of the above scenario is that the network resources are negotiated only after the application level flow properties have been agreed. This prevents the allocation of expensive and shared operating system and network resources until it is clear that the appropriate I/O devices, processing capacity and storage space are available to support the required flow properties. Only after the negotiations have completed, are corresponding stream channels and flows created.

In [Gutfreund95] we consider the integration of multimedia trading and flow filtering into a multi-party connection-management scheme. A salient feature of that approach as compared to the one above, is a connection manager that encapsulate flow property negotiations and thus attempts to hide the fact of flow incompatibilities, from the applications. This we refer to as providing *media format transparency*.

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 point-to-point audio/video connection-management scheme.

A salient feature of our approach is to interpret a type specification as a set of media descriptors. This supports the definition of 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.

We also used the type system to define the properties of a facility supporting a dynamic filter 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.

Clearly standardization is crucial for the solution to the problem of universal interoperability of multimedia applications. In our future work we will therefore address the issue of "standardizing" 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.

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,* Technical Report TR-0303-07-95-380, GTE Laboratories Incorporated, Waltham, MA, 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.

[Gutfreund95] Gutfreund, S., Nicol, J.R., Phuah, V., Eliassen, F., *ATOMS: ATM Support Mechanisms for Advanced Multimedia Applications Research,* Technical Report TR-0297-06-95-380, GTE Laboratories Incorporated, Waltham, MA, June 1995.

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

[OMG95] Object Management Group, *The Common Object Request Broker: Architecture and Specification,* Object Management Group, Framingham, MA 1991.

[vanHalteren95] van Halteren, A, Leydekkers, P., Korte, H., *Specification and Realisation of Stream interfaces for the TINA-DPE* Proc. TINA'95, Melbourne, Australia, February 1995, pp. 299-314.