A TRADING MODEL OF STREAM BINDING SELECTION

F. Eliassen & H.O. Rafaelsen

F. Eliassen, University of Oslo, Dept of Informatics, P.O. Box 1080, 0316 Oslo, Norway,

E-mail frank@ifi.uio.no

H.O. Rafaelsen, University of Tromsø, Dept of Computer Science, 0937 Tromsø, Norway,

E-mail hansr@cs.uit.no

Key words: Multimedia, QoS, Typing, ODP

Abstract: Distributed multimedia information systems require a range of different interaction styles ranging from simple remote operation interaction to complex patterns of interaction involving both discrete and continuous data. The standardized reference model for Open Distributed Processing (ODP) defines a binding model that encapsulates different interaction styles within explicit binding objects. In this paper we introduce the idea of using a trading model for selecting potential binding types. In this scheme a set of binding factories are located based on a specification of the required properties of the binding. The located set of binding factories are all capable of instantiating binding objects with properties conforming to those specified by the client.

1. INTRODUCTION

The notion of stream interface has been proposed as the preferred means to convey media streams in distributed multimedia systems (Coulson et al, 1992). A stream interface consists of a collection of source and/or sink media flows. Stream interfaces have been adopted in the work on Open Distributed Processing (ITU-T Draft Recommendation X.901, 1995) and TINA-DPE (TINA-C, 1995) and more recently by OMG (Object Management Group, 1996).

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: [10.1007/978-0-387-35581-8_35](https://doi.org/10.1007/978-0-387-35581-8_35)

The act of stream binding establishes a logical association between compatible stream interfaces for the purpose of exchanging continuous flows as dictated by the type and direction of the flows. During binding, the interfaces to be bound must be type-checked for compatibility. Informally, type-checking means ensuring that the properties of each source flow are as expected by the corresponding sink flow.

In RM-ODP stream bindings are explicit. This supports direct client control of the binding during its lifetime. In figure 1, a binding object connects four interfaces by means of local bindings that associate the interfaces of the objects with the interfaces of the binding object.

Figure 1. The explicit binding model

A binding type (or template) defines a particular class of binding objects by identifying the type of interfaces which can participate in the binding, the roles they play, and the way behaviour at the various binding interfaces are linked (Lindsey & Linington, 1995). Binding factories are objects that create bindings. In our model, a binding factory is associated to a binding type such that by invoking the factory object's `create` method, a new binding of the associated type is created.

Through the local bindings the binding object receives and delivers information for binding participants according to the causality and type of the bound interfaces. Type checking as explained above, is applied when adding a new interface to the binding. If an object with interface α is offered to fulfil a role β , then the type of α must be compatible with the type of β .

In this paper we present foundational concepts for applications to select binding types and associated binding factories according to their requirements based on a trading model. In this scheme a set of binding factories are located based on a specification of the required properties of the binding. The located set of binding factories have in common that they are all capable of instantiating binding objects with properties conforming to those specified by the client. Such a facility will, for example, allow

evolution of supported binding types in a distributed computing environment without sacrificing support of existing applications. This can be ensured by requiring that a new version of a binding type must *conform* to the older version. Other areas that might benefit from a binding type selection facility as indicated above, are mobile applications that need to trade binding types satisfying their requirements in changing environments, and multimedia databases where the required properties of the binding used to present a query result are generally not known in advance, but rather depends on the specifics of the query (Plagemann et al, 1999).

The remainder of the paper is organized as follows. In section 2, we offer further motivation for our work. Following that, in section 3 we then present the main ideas of our binding framework MBS (Model of Bindings and Streams). Section 4 introduces the notion of binding type and offers a definition of a conformance relationship of binding types. An application of this relationship is illustrated through examples. Section 5 concludes and discusses some further work.

2. TYPE CHECKING AND BINDING OF STREAM INTERFACES

The act of binding includes type-checking of the interfaces to be bound for compatibility. Compatibility rules for stream interfaces, however, have been deemed outside the scope of the RM-ODP standard (ITU-T Draft Recommendation X.901, 1995); the proposed rationale for which is that the optimum form of subtyping will depend on the application. Although this is a valid observation, the advantages of detecting type errors at compile (or bind) time, rather than at run-time, are no less important for stream interfaces than they are for operational interfaces. In the work of TINA-C, the need for a compatibility relationship for stream flows that is more relaxed than equivalence, is recognized, but no definition is offered (TINA-C, 1995). A proposal for audio/video support in CORBA (IONA et al, 1997), also introduces the notion of flow end-point compatibility. This determines whether two flow endpoints are connectable. In their proposal two flow endpoints are compatible if they share a common protocol over which they can transport media, and if they share a common format for media encoding/decoding (e.g. MPEG). Hence QoS parameters are not considered.

Microsoft's ActiveMovie framework (Microsoft ActiveMovie, 1996) also includes the notion of "compatibility negotiation" between "pins" (connection points that carry flows between different processing objects). The subject of this negotiation is data compatibility rather than QoS.

In this work, however, we adopt and extend a generic type model for stream flows and associated type checking rules earlier proposed in (Eliassen & Nicol, 1996) into a generic model of bindings and streams (MBS). In contrast to many of the above approaches, the model of (Eliassen & Nicol, 1996) is open-ended and can in principle support any set of flow parameters. An implementation of the flow type model and examples of its application are described in (Eliassen & Mehus, 1998)

In the flow type model, a flow type is specified by indicating the media type of the flow such as audio or video, its causality (source or sink), and a set of quality attributes such as rate and resolution. Furthermore, an attribute value is specified as a set of "atomic" values. This will in general enhance the chances of successful binding. For example when a sink flow type specifies a set of different names on the video encoding attribute, it actually declares that it can accept flows where the video can have any of the indicated formats. A flow type specification is interpreted as a set of potential stream flow qualities such as data formats and temporal constraints. This supports the definition of a variety of flow type relationships based on set theory. For example, compatibility is determined by computing the set of common flow qualities supported by the two endpoints. Compatibility requires that this set is not empty. If the two endpoints can support more than one common flow quality, a flow quality negotiation protocol may be employed to choose the actual flow quality to be used.

A further feature of the type model of (Eliassen & Nicol, 1996) is that a flow specification may include "individual" requirements to the satisfaction of compatibility such as optional configurations of media elements that may occur in the flow and media element dependencies. In this paper we extend this feature to stream interfaces allowing optional configurations of flows and flow dependencies to be specified. For example, a stream interface modeling an access point to a video conference provider, may exploit this feature to express the alternative audio and video flow configurations and qualities that can be supported. Requiring compatibility ensures that a joining conference participant satisfies at least one of the alternatives supported by the provider.

QML is a recent proposal for a QoS specification language (Frølund & Koistinen, 1998). The semantics of QML is similar to our stream and flow type model but its applicability is demonstrated for operational interfaces only. Other work that considers QoS specifications and/or negotiations includes (Campbell, 1996), (Dini & Hafid, 1997) and (Zinky et al, 1997). However, these works do not provide anything corresponding to a type model of streams and bindings, including type relationships such as subtype, compatibility and conformance.

3. MODEL OF BINDINGS AND STREAMS (MBS)

For the stream abstractions, we have developed a *generic* model called MBS (Model of Bindings and Stream Interfaces). In MBS, a binding object may support a number of stream interfaces. Each interface has an associated role in the binding. As in (Lindsey & Linington, 1995) a role is specified by its name and stream interface type. For example, a multicast video binding would typically support a producer role and a consumer role. In some cases the binding will support operations for adding interfaces to a binding in a named role. A stream interface consists of a collection of source and/or sink media flows.

3.1 Flow Type Model

A flow type is specified by indicating the media type of the flow such as audio or video, its causality (source or sink), and a set of quality attributes such as rate and resolution. Furthermore, an attribute value is specified as a *set of* "atomic" values. For example when a sink flow type specifies a set of different names on the video encoding attribute, it actually declares that it can accept flows where the video can have any of the indicated formats. The following example of an H.261 video flow type features optional playback rates.

```
type VideoPhone = Flow [
  V:Video[encoding:H.261, rate:2..24]];
  A:Audio[encoding:PCM, rate:{8000,16000}]]
(V & A) | A
```

This specification states that a VideoPhone flow consists of two different element types labeled V and A respectively. Each element type includes a declaration of the generic media type such as Video or Audio, and a specific set of attributes, referred to as a media descriptor, specifying quality properties of the element type. The expression (V & A) | A is referred to as the structural constraints of the flow (Eliassen & Mehus, 1998). It specifies legal configurations of flow elements that may occur in an instance of the flow. The above structural constraint indicates that an instance of a flow can consist of either video and audio elements, or audio elements only. Hence we may think of the specification as modelling an adaptable flow endpoint.

A flow type specification is interpreted as a set of potential flow qualities (QI) and flow configuration (SI) that can be produced by a source flow endpoint or is acceptable to a sink flow endpoint. This interpretation of a flow type allows us to define a variety of flow type relationships based on

set theory. The quality interpretation of a flow type is defined as the combination of the interpretation I of each of its element types (for further details see e.g. (Eliassen & Nicol, 1996)).

A flow type M is a *subtype* of the flow type N if both the quality and structural interpretation of M is a subset of the corresponding interpretations of N . We derive that $M = \text{Flow}[A;B]$ is a *strict quality subtype* of $N = \text{Flow}[C;D]$, denoted $M <_q N$, if $I(A) \subseteq I(C)$ and $I(B) \subseteq I(D)$. On the other hand, a *relaxed quality subtype* may support fewer element types than the super type such that, for example, $M = \text{Flow}[A]$ is a relaxed quality subtype of $N = \text{Flow}[B;C]$, denoted $M <\sim_q N$, when $I(A) \subseteq I(B)$. Finally, M is a *structural subtype* of N denoted $M <_s N$, if $SI(M) \subseteq SI(N)$.

Our approach for type checking binding attempts is to require that the source and the sink can at least support one common flow quality. This relationship we refer to as *quality compatibility*. Informally, two flow types are *strict quality compatible* (denoted $<>_q$) if their respective element types have pair wise non-empty set intersection of their respective interpretations. We may for example conclude that $\text{Flow}[A;B] <>_q \text{Flow}[C;D]$ if $I(A) \cap I(C) \neq \emptyset$ and $I(B) \cap I(D) \neq \emptyset$. Two flow types M and N are *relaxed flow quality compatible* (denoted $<\sim>_q$) if a subset of the element descriptors in each of the two flow types, are pair-wise compatible. Thus if $I(A) \cap I(C) \neq \emptyset$, $\text{Flow}[A;B] <\sim>_q \text{Flow}[C;D]$ even if B and D are incompatible, i.e. $I(B) \cap I(D) = \emptyset$. Two flows of type are *structural compatible*, denoted $M <>_s N$, if their structural interpretations have non-empty intersection. Different variants of the compatibility relationship where some variants are weaker than others, are the following:

- i) fully strict compatible, if $M <>_q N$ and $M <_s N$.
- ii) partially strict compatible, if $M <>_q N$ and $M <>_s N$.
- iii) fully relaxed compatible, if $M <\sim>_q N$ and $M <_s N$.
- iv) partially relaxed compatible, if $M <\sim>_q N$ and $M <>_s N$.

3.2 Stream Type Model

In MBS the notion of configuration constraint allows for the specification of individual requirements to the satisfaction of compatibility for stream interfaces in terms of alternative combinations of flows that may be configured in a stream binding. A stream configuration constraint is written as a structural constraint over flow labels. The set of alternative configurations of flows of a stream interface is referred to as its structural intersection SI .

Two stream interfaces S and T are *compatible*, denoted $S <> T$, if S and T have a common legal configuration such that the flows in these

configurations of **S** and **T** are pairwise compatible. Consider, for example, the following stream interface type of a video conference binding type

```
type videoConfProducer = stream [
  a : sink flow [
    a1: Audio[encoding:PCMA,
              rate:{8000,16000}]]];
  v : sink flow [
    v1: Video[encoding:H.261,
              rate:(2..24)]]]
configurations a|(a&v) ] //end stream
```

and the interface type `audioTalk` offered by a potential participant of the binding

```
type audioTalk = stream [
  a : source flow [
    a1: Audio[encoding:{PCMA,GSM},
              rate: 8000]]; //end flow
configurations a ] //end stream
```

The above interface types are compatible.

The MBS *conformance rules* express conditions for substitutability of stream interfaces. If the stream interface **T** *conforms* to the stream interface **S**, then **S** may be replaced transparently by **T**. A stream interface **T** conforms to a stream interface **S** if and only if for every flow of **S** there is a corresponding flow in **T** such that the flow of **S** is a subtype of the flow of **T** and $SI(S) \subseteq SI(T)$.

The following example illustrates a case where a video source `mpgSource` is upgraded to support additional playback rates, video encodings and audio, all encapsulated in the stream interface `mpg_mjpgSource` such that `mpg_mjpgSource` conforms to `mpgSource`.

```
type mpgSource = stream [
  mpgFlow : source flow[
    h: Video[encoding:mpeg,rate:{20,25}]]
configurations mpgFlow ] //end stream

type mpg_mjpgSource = stream [
  mpgFlow : source flow [
    h: Video[encoding:mpeg,rate:{20,25,30}]]];
  mjpgFlow : source flow [
    hj: Video[encoding:mjpeg,rate:{20,25,30};
    au: Audio[encoding:{PCMA,GSM},rate:8000]]
configurations mpgFlow|mjpgFlow ] //end stream
```

4. SELECTION OF BINDING TYPE

In this section we present an approach for selecting binding types from binding type requirement specifications. Our approach for specifying binding types is similar to the RIVUS template language (Lindsay & Linington, 1995), the main difference being that binding requirements are specified using the stream interface type model introduced in this paper.

4.1 Specifying binding types

A binding type is defined as a 5-tuple $\langle T, P, M, \Delta, E \rangle$ where T denotes a set of role types, P a set of roles, M a set role matching requirements (one for each role), Δ a set of role causalities, and E a set of role cardinality requirements.

A *role type* τ is defined as a set of stream interface types, i.e. $\tau = \{T_1, \dots, T_n\}$. A *role* defines binding object roles and is specified as a role name and a role type, $r: \tau$. For example, a video conference binding type could define the roles `talk` and `listen` where the role `talk` could be of the role type `{videoConfProducer}` (c.f. section 3.2).

Role matching requirements specify for each role of the binding type the kind of type matching required when an interface is offered to fulfil the role. The kind of matching that can be specified is either a subtype or a compatibility relationship. Thus we model role matching requirements as a set of pairs $\{\langle r_1, m_1 \rangle, \dots, \langle r_n, m_n \rangle\}$ where r_i is a role name and m_i is a match kind.

Binding behaviour defines *causalities* between instances of roles. We model role causality as a tuple $\langle C, r_1, r_2, m \rangle$ where C specifies a causality option, r_1 and r_2 are roles, and m specifies whether conversion between r_1 and r_2 is supported by the binding. Conversion is supported if $m = \text{conv}$, otherwise $m = \text{no_conv}$. Conversion is required if the roles r_1 and r_2 have incompatible roles types, and may be required in those cases where alternative behaviour is specified at the corresponding interfaces (e.g. alternative encodings) and the chosen behaviour for r_1 is allowed to be incompatible with the chosen behaviour for r_2 . For example, if the type of r_1 and r_2 is both `video`, and the type `video` offers both `mpeg` encoding and `mjpeg` encoding, and r_1 chooses a configuration with `mpeg`, and r_2 chooses a configuration with `mjpeg`, then a conversion between `mpeg` and `mjpeg` is needed. This might, for example, be realised as a suitable transcoder running within an active network.

As in (Lindsay & Linington, 1995) we define three options for how roles can be mapped together. Specifying $\langle \text{ONE-ONE}, r_1, r_2, m \rangle$ means that the

binding object creates a one to one mapping between a single instance of role r_1 and a single instance of role r_2 , while $\langle \text{ONE-MANY}, r_1, r_2, m \rangle$ means that the binding object creates a one to many mapping between a single instance of role r_1 and all instances of role r_2 . Finally, $\langle \text{MANY-MANY}, r_1, r_2, m \rangle$ means that the binding object creates a mapping between all instances of role r_1 and r_2 .

Role cardinality is a specification of the number of a particular role the binding object can support (Lindsey & Linington, 1995). It is modelled as a pair $\langle r, i \rangle$ where r is a role and i is a range where the minimum value states the number of the role that is needed for the binding object to make sense, while the maximum states the largest number of the role the binding object is willing to support.

$\langle \text{talk}, 2..10 \rangle$

4.2 Binding type conformance

An application selects a binding type by stating binding type requirements to a trader that compares the requirements to binding type offers. A binding type requirement specification is with one exception only, identical to a definition of a binding type as outlined above, while a binding type offer is simply a binding type specification. Selection is based on a conformance relationship between binding type requirements and binding type offers. The result of the selection is the identification of a set of binding factories that are all capable of instantiating binding objects with properties conforming to those specified by the client. A conformance relationship for binding types must be based on conformance of stream interfaces and notions of role matching, role causality, and role cardinality satisfaction.

While a role matching requirement of a binding type offer is specified as a pair $\langle r, m \rangle$, the role matching requirement of a binding type requirement is specified as a triple $\langle r, m, \sigma \rangle$ where σ indicates whether a stricter role matching requirement than m is acceptable ($\sigma = \text{narrow}$) or not ($\sigma = \text{no_narrow}$). It is easy to show that strict subtype logically implies ($\Rightarrow$) all other match kinds, relaxed subtype logically implies fully and partially relaxed compatibility, full compatibility logically implies partial compatibility, and strict compatibility logically implies relaxed compatibility. Thus, $\langle r, \text{relaxed_compatibility}, \text{narrow} \rangle$ is satisfied by $\langle r, \text{full_compatibility} \rangle$.

Definition 1 (*role matching satisfaction*) A role matching requirement $\langle r_1, m_1 \rangle$ of a binding type offer, satisfies a role matching requirement $\langle r_2, m_2, \sigma \rangle$ of a binding type requirement if and only if $m_1 = m_2$, or $\sigma = \text{narrow}$ and $m_1 \Rightarrow m_2$. $\square$

Definition 2 (*role causality satisfaction*) A role causality $\langle C, r_1, r_2, m \rangle$ is satisfied by a role causality $\langle C', s_1, s_2, n \rangle$ if and only if $C = C'$, s_1 conforms to r_1 , s_2 conforms to r_2 , and if $m \neq n$, then $n = \text{conv}$. $\square$

Definition 3 (*role type conformance*) A role type $\tau = \{T_1, \dots, T_n\}$ conforms to a role type $\sigma = \{S_1, \dots, S_n\}$ if and only if there exists a bijection β between τ and σ such that for all $(T_i, S_j) \in \beta$, T_i conforms to S_j . $\square$

Definition 4 (*binding type conformance*) A binding type offer $B_1 = \langle T_1, P_1, M_1, \Delta_1, E_1 \rangle$ conforms to a binding type requirement $B_2 = \langle T_2, P_2, M_2, \Delta_2, E_2 \rangle$ if and only if there exists a bijection β between the sets of role causalities Δ_1 and Δ_2 such that for all $(\delta, \varepsilon) \in \beta$ with $\delta = \langle C_1, r_1, s_1, m_1 \rangle$ and $\varepsilon = \langle C_2, r_2, s_2, m_2 \rangle$, δ satisfies ε , and the role cardinality requirements $\langle r_1, i_1 \rangle$ and $\langle s_1, j_1 \rangle$ in E_1 satisfies the corresponding role cardinality requirements $\langle r_2, i_2 \rangle$ and $\langle s_2, j_2 \rangle$ in E_2 such that $i_2 \subseteq i_1$ and $j_2 \subseteq j_1$, and the role matching requirements $\langle r_1, m_1 \rangle$ and $\langle s_1, n_1 \rangle$ in M_1 satisfies the corresponding role matching requirements $\langle r_2, m_2, \sigma_2 \rangle$ and $\langle s_2, n_2, \mu_2 \rangle$ in M_2 . $\square$

Example 1. The following is a simple specification of a binding type requirement, for an audio conference binding. We will later refer to this binding type by the name `audioConf`. We define two role types `audioConfProducer` = {audioTalk} and `audioConfConsumer` = {audioListen} where

```
type audioTalk = stream [
  a : sink flow [
    a1: Audio[encoding:{PCMA},rate: 8000]]
  configurations a ] //end stream

type audioListen = stream [
  a : source flow [
    a1: Audio[encoding:{PCMA},rate: 8000]]
  configurations a ] //end stream
```

The corresponding binding roles are

```
talk : audioConfProducer
listen : audioConfConsumer
```

Note that the causalities of the flows are specified as they are provided by the binding, i.e. a source flow of an audio conference participant (a talker) must locally bind to a corresponding sink flow offered by the binding.

The role matching requirements are

```
<talk,partially_relaxed,narrow>
<listen,partially_relaxed,narrow>,
```

while the role causality required by the audio conference application is

```
<MANY-MANY,talk,listen,no_conv>,
```

and the role cardinality requirements are <talk,2..6> and <listen,2..6>.

A binding type offer that satisfies the above binding type requirement is specified below. We will later refer to this offer by the name AVConf. The offer defines two role types AVConfProducer={AVTalk} and AVConfConsumer={AVListen} where

```
type AVTalk = stream [
  a : sink flow [
    a1: Audio[encoding:{PCMA,GSM},
      rate:{8000,16000}]]];
  v : sink flow [
    v1:Video[encoding:H.261,
      rate:(2..24)]]]
  configurations a|(a&v) ] //end stream

type AVListen = stream [
  a : source flow [
    a1: Audio[encoding:{PCMA,GSM},
      rate:{8000,16000}]] ];
  v : source flow [
    v1: Video[encoding:H.261,
      rate:(2..24)]]]
  configurations a|(a&v) ] //end stream
```

The binding roles of the offer are

```
gen : AVConfProducer
rcv : AVConfConsumer
```

The role matching requirements are

```
<gen,fully_strict_compatible>
<rcv,fully_strict_compatible>,
```

while the role causality requirement of the binding type offer is

```
<MANY-MANY,gen,rcv,conv>,
```

and the role cardinality requirements are <gen,2..20> and <rcv,0..20>.

It can easily be shown that the binding type offer satisfies the binding type requirement according to definition 4. We basically need to show that

the role causality requirements of the binding type offer, <MANY-MANY,gen,rcv,conv>, satisfies to role causality requirements of the binding type requirement, <MANY-MANY,talk,listen,conv>, and that role cardinality and role matching requirements of corresponding roles are satisfied. $\square$

Below we give a further example of how binding types and stream interface types can be used as a foundation for creating explicit bindings.

Example 2. Suppose two users, User_A and User_B, want to engage in a teleconference. Their capabilities are given by their stream interface specifications.

User_A supports two stream interface types: videoConfProducer, specified in section 3.2; and videoConfConsumer specified below.

```
type videoConfConsumer = stream [
  a : sink flow [
    al: Audio[encoding:PCMA, rate:{8000,16000}]]];
  v : sink flow [
    vl:Video[encoding:H.261, rate:2..24]]
  configurations a|(a&v) ] //end stream
```

User_B supports the stream interface types: audioTalk, specified in section 3.2; and audioListen specified below.

```
type audioListen = stream [
  a : sink flow [
    al: Audio[encoding:{PCMA, GSM}, rate:{8000}]]
  configurations a ] //end stream
```

Figure 2. A conference binding

Suppose User_B initiates the teleconference by requesting a reference to an appropriate binding factory from the binding trader. In this case we assume that User_B generates the binding type requirement, `audioConf`, as specified in example 1, by comparing the interface types supported by both users.

For the sake of the example we further assume that the distributed processing environment in this case does not offer a binding exactly as specified by User_B. However, the environment offers a binding with the properties of `AVconf`, as specified in example 1.

Since the offered binding `AVconf`, conforms to the requested `audioConf`, the offered binding can be used. Figure 2 shows how the specifications of User_A and User_B are used to create a explicit binding. It is possible for both users to attach to the binding, since the binding provides compatible stream interfaces. New participants might join the conference by invoking operations at the bindings control interface. The binding will check that the new participants interfaces are compatible with the offered interfaces of the binding. □

5. CONCLUSIONS AND FUTURE WORK

In this paper we introduced foundational concepts supporting mechanisms for selecting appropriate explicit stream bindings based on statements of binding type requirements provided by the application. We introduced the idea of using a trading model for selecting potential binding types, and showed how an earlier proposed type model for stream flow can be extended to support stream interfaces and binding type selection based on a notion of binding type conformance.

In our current work we focus on extending the type model for supporting the situation where binding objects specify alternative behaviour at its supported interfaces and it is needed to negotiate the actual interface behaviour to be used. Our approach is to introduce the notion of *policy specifications* that can be associated to each flow of a stream interface. A policy specification effectively specifies an order on the interpretation of a flow type. This ordering can, for example, be used to select QoS parameters for the flow based on a given resource availability, and for adapting the quality of the flow by taking the flow interpretation order as the specification of the preferred degradation path. The results of this work will be reported elsewhere.

REFERENCES

- Campbell, T. (1996). A Quality of Service Architecture. *PhD Thesis*, Lancaster University
- Coulson, G., Blair G. S., Stefani, J. B., , Horn, F., Hazard, L. (1992). Supporting the Real-Time Requirements of Continuous Media in Open Distributed Processing. *Technical Report MPG-92-35*, Lancaster University.
- Dini, P., Hafid, A. (1997). Towards Automatic Trading of QoS Parameters in Multimedia Distributed Applications, *In proceedings of IEEE/IFIP ICODP/ICDP Conference*, Toronto, Canada, 166 - 179
- Frølund,, S., Koistinen, J. (1998). Quality-of-Service Specification in Distributed Object Systems, *Distributed Systems Engineering Journal*, Vol.5, No.4
- Eliassen, F., Nicol, J. R. (1996). Supporting Interoperation of Continuous Media Objects. *Theory and Practice of Object System*, Vol.2, No.2, Wiley, 95-117.
- Eliassen, F., Mehus, S. (1998). Type Checking Stream Flow Endpoints. *Middleware '98*, The Lake District, England, 16-18 Sept, Chapman & Hall, 305 - 322.
- Lindsey, D., Linington, P.F. (1995). RIVUS: A Stream Template Language for Capturing Multimedia Requirements, *Lecture Notes in Computer Science (LNCS 1052)*, Springer Verlag, 259 - 277
- Microsoft. (1996). Microsoft ActiveMovie: Software Development Kit, Beta Release, June 1996.
- ITU-T X.901 | ISO/IEC 10746-1. (1995). ODP Reference Model Part 1: Overview. *Draft International Standard*.
- IONA Technologies, Pic, Lucent Technologies, Inc, Siemens-Nixdorf, AG. (1997). Control and Management of A/V Streams Request for Proposal. OMG RFP Submission, *OMG Document: telecom/97-05-07*.
- Object Management Group. (1996). Control and Management of A/V Streams Request for Proposal. *OMG Document: telecom/96-08-01*.
- Plagemann, T., Eliassen, F., Göbel, V., Kristensen, T., Rafaelsen, H., Adaptive QoS Aware Binding of Persistent Multimedia Objects, *In proceedings of International Symposium on Distributed Objects and Applications (DOA'99)*, Edinburgh 5-6 sept 1999 (to appear)
- TINA-C. (1995). TINA Object-Definition Language, Version 1.3. *TINA-C Deliverable*.
- Zinky, A., Bakken, D.E., Schantz, R.D. (1997). Architectural Support for Quality-of-Service for CORBA Objects, *Theory and Practice of Object Systems*, Vol.3, No.1, Wiley