

# Dynamically Solving Media Incompatibility

Hans Ole Rafaelsen  
Dept. of Computer Science  
University of Tromsø  
Norway  
hansr@cs.uit.no

Frank Eliassen  
Simula Research Laboratory  
P.O.Box 1080 Blindern  
0316 Oslo, Norway  
frank@simula.no

## Abstract

We propose the Gateway Description Language (GDL) used to specify properties of media gateways. GDL describes the interfaces of a gateway, and the dependency between them. Gateways descriptions in GDL are used within a trading service to locate gateways, based on the properties of the gateway. This service can be used to automatically resolve incompatibly between heterogeneous participants in a multimedia binding. Based on the requirements of GDL and the trading service, a prototype is implemented using the trading service of CORBA, in order to evaluate the suitability of the CORBA trader for trading media gateways. The experiment shows that the CORBA trader has some short comings. This makes it necessary to introduce other entities to work in conjunction with the trader, to dynamically locate proper gateways for multimedia bindings.

## 1 Introduction

The increase in multimedia enabled computers and mobile devices with multimedia capabilities, high speed networks and mobile networks is drastically increasing the use of distributed multimedia applications. Such applications range from distance education applications for teaching and training to integrated command and control systems for naval defense vehicles.

The large number of different devices and applications results in a vast number of different media types. Media types represent both the encoding used and the values for the various QoS-dimensions. Getting these heterogeneous media types to interoperate will be an important challenge which next generation multimedia enabled applications have to overcome.

The increase in different media types, and the way different applications interact with each other, makes it necessary to determine whether two applications can interoperate. This has to be done at run-time. Furthermore, if two applications are determined to be unable to interoperate directly with each other, necessary steps should be taken in order to try working around this compatibility problem.

For example, if two users with applications supporting different media types want to engage in a video conference, they might fail due to lack of interoperability between the different media types. What is needed in this situation is first of all,

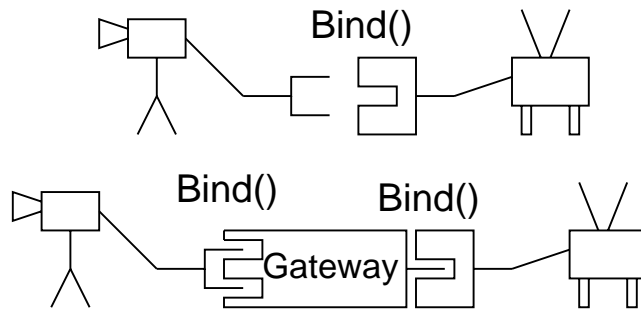


Figure 1: Gateway to adapt from two incompatible formats

to be able to actually determine that the properties of the media type produced by the sender is not compatible with the type expected by the receiver. Once that has been determined, the system should automatically try to find a *media gateway* (also called filter) which is able to translate between the two media types.

Media gateways (hereafter called gateways) provide a mechanism which can be introduced to dynamically solve heterogeneity within a binding framework. The purpose of a gateway is to transform from one media type to another. Figure 1 gives an illustration of the principle of usage of media gateways. The type (media format) from the camera is of a type which the TV is not capable of displaying. This can be solved by inserting a gateway between them. The gateway translates from the type produced by the sender on its *source interface*, to a type expected by the receiver on its *sink interface*.

We propose a model to automatically determine compatibility between two interfaces. If two interfaces are compatible, they are capable of binding to each other. This compatibility test is done dynamically, and is done when the interfaces tries to create a binding at run-time. A framework for doing this has been presented previously [5] [3] [4]. An extension to it has been proposed in [12], where the specific media type to use is determined based on *policies*, given that the two interfaces have more than one media type configuration in common.

The binding framework is based on the concept of *open bindings* [6]. An open binding is a composite distributed object, used to connect multiple interfaces across a distributed computing environment. An open binding usually has multiple levels of composition, which means that a binding can be composed of lower level bindings. This provides multiple levels of abstractions and support reuse of binding specifications. Figure 2 shows a open binding. The composite binding is made of several components and inner bindings. This particular binding has two gateway components,  $G_1$  and  $G_2$ , and three inner one to many bindings.

In this paper we investigate the requirements needed to specify the properties of gateways. We propose a language for specifying media gateways, and a method to search for gateway offers within the run-time environment. Based on the requirements for specifying and locating gateways, we describe an implementation of this based on the CORBA trading service [7]. The solution used in the CORBA implementation is evaluated toward the proposed requirements. Also a performance evaluation of the implementation is presented.

In the next section we present our media gateway framework. Thereafter we present a gateway trader in section 3. The CORBA prototype is presented and

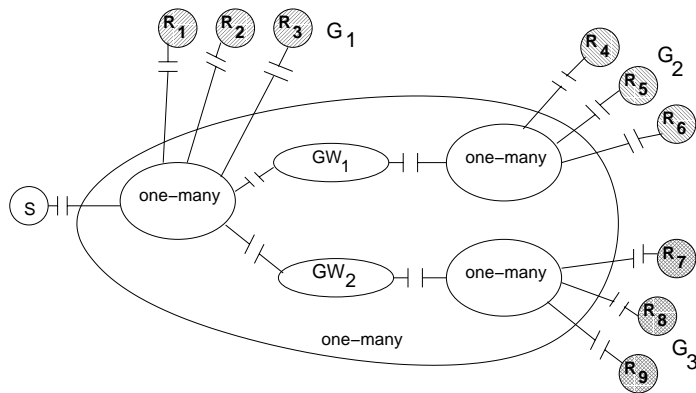


Figure 2: The open binding model

evaluated in section 4 and a small example of its usage is given. Section 5 presents related work. The conclusion and an outline for further work is finally given in section 6.

## 2 Gateway Framework

In order to introduce gateways to automatically, at run-time, solve the heterogeneity gap between interfaces in a binding, it becomes necessary to have a way to reason about gateways. There has to be a way to describe the properties of a given gateway.

The specification for gateway capabilities are based on the work for describing capabilities of flow interfaces [5] [4]. In this work, a type system is developed in order to reason about properties of flow interfaces. In particular, whether interfaces can be bound to each other. As part of this work, the *Flow Interface Description Language* (FIDL) was developed. This language enables an application programmer to specify the interface characteristics of media flow endpoints. This is also referred to as the type of the interface. In addition, algorithms were developed to determine if interfaces specified in FIDL are compatible such that they can be meaningfully bound to each other. Interfaces are made up of flow entities. The properties of the entities are given by a set of attributes. Attributes have name and values. Values can be a single value, a set of values or ranges of values. The language also has a notation of constraints which specifies how elements can be combined in the interface. That is, constraint expressions specify which constellations of element types are allowed to appear at a flow endpoint. They are expressed as logical *and* and *or* expressions between elements. Streams in FIDL are sets of directed flows.

Program 1 shows an example of two interfaces, specified in FIDL. `IF_A` is a source stream of flow named `flowExample`. Entity `v` is of generic type `video`, and contains two attributes. Attribute `encoding` has a single value, while `samplerate` contains a set of values. The constraint `a & v` requires that both entities are presented at the interface.

### 2.1 Gateway Specification

Given incompatible interface specifications of two interfaces to be bound, there should be a way to specify the behavior of gateways in order to automatically de-

---

**Program 1** FIDL description of two interfaces

---

```
stream IF_A {
  source flow flowExample {
    video v {
      encoding = "MJPEG";
      samplerate = {20, 25, 30};
    };
    audio a {
      encoding = "GSM";
      samplerate = 8000;
      channels = {1};
    };
    constraint a & v;
  }; // flow
}; // stream

stream IF_B {
  sink flow flowExample {
    video v {
      encoding = "H261";
      samplerate = 29.97;
    };
    audio a {
      encoding = "G721";
      samplerate = 8000.0;
      channels = {1, 2};
    };
    constraint a & v;
  }; // flow
}; // stream
```

---

termine if a given gateway is suitable to be inserted as a converter between the participants. We call this a *gateway specification*, and its goal is to capture the behavior of gateways in such a way that we can *algorithmically* reason about their suitability. The specification specifies the input and output interface of the gateway, and the casual dependency between those interfaces. That is, the configuration of the output interface depends on the configuration of the input interface, and the *functionality* of the gateway.

Formally a gateway can be described as a function which take a set of streams as input, and produces another set of streams as output.  $f(s) \rightarrow s'$ , where the streams  $s$  and  $s'$  represents set of properties  $p_1, p_2, \dots, p_n$  and  $p'_1, p'_2, \dots, p'_m$  respectively.

## 2.2 Gateway Definition Language

We have developed a new language, *Gateway Definition Language* (GDL), to describe gateways. It is an extension of FIDL. It consist of two parts, the *interface description part* and the *dependency part*. The interface description part is a FIDL description of the two interfaces. In it simplest case, a gateway in GDL is described as two interfaces described in FIDL. This is for the cases where there is no dependency between the configuration of the sink and source interface. Thus, the dependency part will be empty. In cases where there are dependencies, the values of properties of the source interface are represented as functions of the properties of the sink interface it depends on. When determining legal configurations of the gateway, descriptions in the dependency part takes precedence over descriptions in the interface description part, when evaluating the configuration of the source interface.

## 2.3 Dependency functions

A dependency function is a function which can take one or more properties as parameters, and returns one or more properties as result. This can be expressed as,  $f(p_1, p_2, \dots, p_n) \rightarrow p'_1, p'_2, \dots, p'_m$ . Note that there is no dependency between the number of parameters given to a function, and the number of results returned. In

addition to specifying dependencies between attributes, dependency functions are also used to specify dependencies between flow elements.

Program 2 gives examples of dependency functions which reduces the spacial resolution to a quarter.

---

**Program 2** An example of a spacial property function.

---

```
size_t, size_t quarter_size(size_t x, size_t y) →  
    x_half := x / 2; y_half := y / 2; return x_half, y_half;
```

---

## 2.4 Discussion

Even though the gateways we consider in this paper is limited to provide a single sink and a single source interface, they can be used to support one to many bindings with heterogeneous receivers (see figure 2). This binding consist of one sender interface, denoted  $S$ . There are nine receiver interfaces  $R_1, \dots, R_9$  portioned into three groups  $G_1, G_2$  and  $G_3$ . Receivers in each group are assumed to have mutually compatible interfaces, while their interfaces are incompatible with those of all receivers in other groups. Hence the groups must necessarily be disjoint. Furthermore, we assume that the interfaces of the receivers of group  $G_1$  are compatible with  $S$ , thus the flow from  $S$  can be bound to the receivers of  $G_1$  without the need for transcoding. For receivers in  $G_2$ , we assume the interfaces are incompatible with  $S$ . Hence the flow from from  $S$  to the receivers of  $G_1$  must be transcoded. This is achieved with the gateway  $GW_1$ . Similarly receivers in  $G_3$  needs transcoding. This is achieved with the gateway  $GW_2$ . Many to many bindings can be resolved as a set of one-to many bindings. More details of the binding framework is described in [12].

## 3 Gateway Trader

To take advantage of gateways in a binding framework, there has to be a way to locate them. The idea is to have a *gateway trader* helping to *dynamically* resolve heterogeneity. Trading allows the application to locate a gateway based on the properties of the gateway, without knowing the exact gateway. Thus, gateways are located based on what is required by the gateway, and not by specifying a specific gateway.

Services announcements provided by gateways are stored as GDL descriptions in the trader. The trader searches for gateways by trying to match the source interface of the sender to the sink interface of the gateway, and the sink interface of the receiver to the source interface of the gateway. Once a match has been found, the trader makes sure that the dependency between the sink and source interface of the gateway holds. This is done by testing that for specific configurations of the sink, the resulting configuration of the source, is still of a type which the receiver is accepting.

Within our binding framework[11], a *gateway trader* is responsible for helping the *binding factory* to locate proper gateways. In our framework gateways are additional services, which the binding has to be bound through. Gateways are offered as services within the run-time environment.

## 4 CORBA Implementation

A prototype has been implemented using the CORBA trader [7] as the gateway trader. In order to use the trader, GDL specifications of filters have to be translated into CORBA trader service offers.

### 4.1 CORBA Trading Service

Below we give a short overview of the CORBA trader with regard to what is of importance to gateway trading.

The CORBA trader stores advertisements for services, called *service offer*. In our case the service will be the gateways. The advertisement is exported to the trader by the *exporter*. Description of the service inside a service offer is provided by a number of name-value pairs called *properties*. Searching the trader for a service provider that meets certain criteria is known as *import*, and will be conducted by the binding factory in our binding framework.

Service offers have a type, known as the *service type*. In addition to the service type which determines which properties are used to describe a particular service offer, the object reference to the service provider that is stored in each service offer has an *IDL interface type*. This will be the gateway factory type in our case.

The properties in the service type, can be of any valid IDL types, also user-defined. However, the query language used to search for offers only support simple IDL types. Constraint expressions, also called queries, are Boolean expressions over the property values of service offers. To select a service offer to be returned, the importer specifies a constraint using the constraint language. As part of the query, the importer can use *policies* to specify which properties should be returned as part of the service offer.

### 4.2 GDL to CORBA Service Offers

Since the trader does not allow queries against complex data types, structural constructs of a gateway given in the GDL specification have to be preserved through enforcement of naming rules of the trading *service type*.

A GDL interface specification *attribute* value is transformed into a CORBA property value by the following rules. Limitations due to the lack of structuring of attributes in trader constraint language has been solved by incorporating the GDL structure into the naming convention of the property names. An attribute in the constraint language must have the following fields: `<direction>` `<element_name>` `<attribute_name>`, when represented as CORBA properties.

Attributes containing single values or set values are stored as CORBA sequences. Range values, set of range values and values of a type containing an implicit ordering, i.e. numbers, are represented with the use of two properties. One of the values represents the minimum value, while the other represents the maximum value. They are encoded as `<attribute>_min` and `<attribute>_max` respectively.

We have not found a good way to express the constellations which results from the constraints, as CORBA offers. The best solution found so far, is to export the different combinations as different offers. This will have the consequence of having to store more offers at the trader, which will require that more gateways have to be searched when importing offers.

The dependency part and dependency functions are stored as sequences of strings. That is, we use two properties to store this information. One to hold the text which represent the dependency part, and the dependency functions are stored by their code in the the other sequence. Also, a property of type sequence of string, is used to store the whole interface description part.

All gateway factories support the same operational *IDL interface type Gateway Factory*, that can be accessed by a binding factory after it has got a reference to it. Among the operations supported by a gateway factory are operations to initiate a gateway, and to bind the gateway to the stream endpoints.

### 4.3 FIDL to CORBA Query

To search for a gateway offer, the binding factory has to transform the FIDL specification of the interfaces into a *query* in the CORBA constraint language. Below we outline an algorithm for this transformation.

There are two types of attributes describing the registered gateways. The first type of attributes are represented as a CORBA property sequence of values. The *encoding* fields in example 3 are examples of such attributes. A search against these fields will be to find the values in the sequence. The other type of attributes are made up of two properties. One for the minimum value the attribute can have and the other for the attribute maximum value. Together they represent the range of values of the attribute. A query against this range value will test if the value range for the attribute of the endpoint interface intersects with the range of the attribute of the gateway.

The CORBA trader *policy* parameter is used to instruct the trader to return three properties as part of the offer references. These are the dependency functions, the dependency part and the interface description part of GDL.

#### 4.3.1 Discussion

In GDL entities are of a *generic type*, i.e. video or audio. These generic types are used as part of the name when converting the attribute name to a CORBA trader property name. This preserves the structuring of attribute names. The approach has one drawback. If the constraints expression permits interfaces with several entities of the same generic type to appear at the interface, then the same property name might be tried exported to the trader multiple times. This will result in a duplicate property name error. These situations are however not likely to take place. It requires that the interfaces have two appearance of the same generic type, e.g. video, and that both has to be present at the concrete configuration of the interface.

The inaccuracy resulting from transformation of range values is dealt with by making sure that trader queries will find all possible offers that conforms to requirements at the interface level. However, dependency relationships are not considered by the trader. Hence offers that are potentially “false positives” might be included in the set of conforming offers. This is dealt with by the binding factory that gets the whole GDL specification and dependency functions returned from the CORBA trader as part of the offers. This information can then be used to determine if the offer also conforms with respect to the dependency relation, before an attempt to create the binding is made. Thus, the responsibility is lifted from the trader to the

binding factory, on the cost of having a more complex binding factory, and requires that we store additional information.

---

**Program 3** Example gateway description in GDL.

---

```
filter FilterExample {
  stream snkExample { sink flow flowExample {
    video v { encoding = "MJPEG"; samplerate = (15, 20, 25); };
    audio a { encoding = "GSM"; samplerate = {8000.0, 16000.0,
      32000.0}; channels = {1}; }; constraint v & a;
  }; }; // Stream

  stream srcExample { source flow flowExample {
    video v1 { encoding = "MPEG"; samplerate = (1.0, 10.0)
      + 20.0; };
    video v2 { encoding = "H261"; samplerate = 29.97; };
    audio a { encoding = "G721"; channels = {1, 2};
      samplerate = {8000.0, 16000.0}; };
    constraint (v1 | v2) & a; }; }; // stream
}; // filter
```

---

## 4.4 A Gateway Example

In this section we present an example that show how a gateway description in GDL is transformed to a representation in the CORBA service type description.

The gateway is described by two interfaces. One for the sink and one for the source. Program 3 gives an example of a gateway description in GDL. This specific gateway does not have any dependencies between the interfaces.

Since the GDL gateway descriptions constraint statement allows for two alternative configurations of the gateway, they are exported as two gateways to the trader. Program 4 shows the exported gateways offers as CORBA property types.

Program 1 gives the FIDL description of two interfaces which a binding factory tries to bind. The interfaces are incompatible and a query will be made to search for a gateway, see program 5.

## 4.5 Performance

We evaluated the performance of the trader with regard to how it scaled with respect to gateway complexity and number of gateways registered in the trader. The experiments were done by changing one of the factors at the time, to evaluate its inflection on performance, and also how the combination inflected on the performance. The complexity of a gateway is given by the number of attributes the gateway contains. In our experiment we used gateways containing 10, 50 and 100 attributes. For each of these groups of gateways the trader was populated with 10, 30, 50 and 100 gateway offers. We assume that most filter specifications will have number of attributes within this range, and that the number of offers within a trading domain will be less than a hundred. The gateways stored at the trader are of two categories. One category is the set of gateways that has properties conforming to properties we are

---

**Program 4** CORBA service type properties of gateway export 1 and 2.

---

export 1	export 2
snk_v_encoding = MJPEG	snk_v_encoding = MJPEG
snk_v_samplerate_min = 15.0	snk_v_samplerate_min = 15.0
snk_v_samplerate_max = 25.0	snk_v_samplerate_max = 25.0
snk_a_encoding = GSM	snk_a_encoding = GSM
snk_a_channels_min = 1	snk_a_channels_min = 1
snk_a_channels_max = 1	snk_a_channels_max = 1
snk_a_samplerate_min = 8000.0	snk_a_samplerate_min = 8000.0
snk_a_samplerate_max = 32000.0	snk_a_samplerate_max = 32000.0
src_v_encoding = MPEG	src_v_encoding = H261
src_v_samplerate_min = 1.0	src_v_samplerate_min = 29.97
src_v_samplerate_max = 20.0	src_v_samplerate_max = 29.97
src_a_encoding = G721	src_a_encoding = G721
src_a_channels_min = 1	src_a_channels_min = 1
src_a_channels_max = 2	src_a_channels_max = 2
src_a_samplerate_min = 8000.0	src_a_samplerate_min = 8000.0
src_a_samplerate_max = 16000.0	src_a_samplerate_max = 16000.0

---

---

**Program 5** CORBA query to search for a gateway between the two endpoints

---

```
(('MJPEG' in snk_v_encoding) and ((snk_v_samplerate_min
<= 30) and (snk_v_samplerate_max >= 20)) and
('GSM' in snk_a_encoding) and ((snk_a_samplerate_min
<= 8000.0) and (snk_a_samplerate_max >= 8000.0)) and ((
snk_a_channels_min <= 1) and (snk_a_channels_max >= 1)))
and
(('H261' in src_v_encoding) and ((src_v_samplerate_min
<= 29.97) and (src_v_samplerate_max >= 29.96)) and ('G721'
in src_a_encoding) and ((src_a_samplerate_min <= 8000.0)
and (src_a_samplerate_max >= 8000.0 )) and ((
src_a_channels_min <= 2) and (src_a_channels_max >= 1)))
```

---

searching for in one of the searches, and the rest of the gateways have properties that will not give a match for any of the searches.

We conducted two types of searches. The first was searching for a gateway with properties not found in the trader. The other was to search for the gateway which is known to give a match for one of the offers. The two types of search were done locally on the same computer as the trader, and remotely from an other computer on the LAN. The time recorded is the time it took calling the `CosTranding::Lookup::query()` method, and does not take into account additional overhead like locating the trader.

The ORB used was TAO version 1.1 [13], OS used Solaris 5.7 running on a Sun Ultra 60 with 1 GB RAM. The computer used to do the remote evaluations was a Sun Ultra 5 with 128 MB of ram. The two computers are connected via a 100Mbit/s Ethernet. Under the experiments, the background load on the computers and net where low.

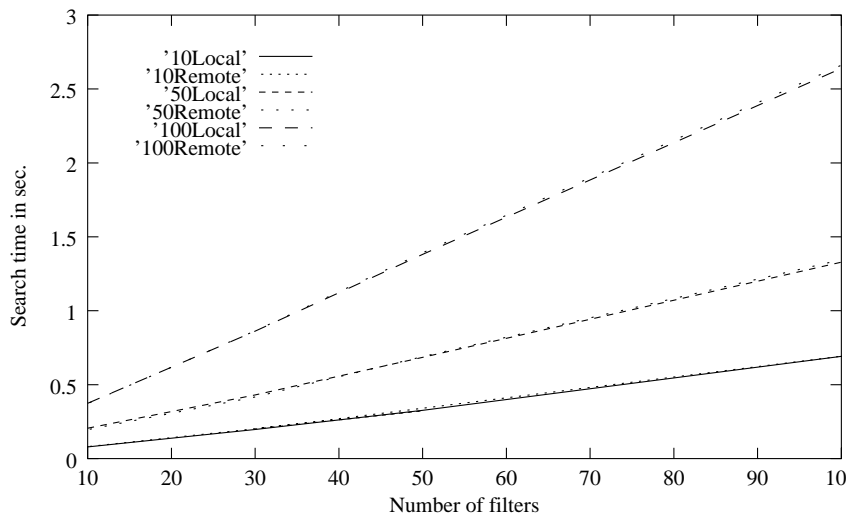


Figure 3: Time spent to search for gateways

From the results it can be seen that the search time rises approximately linearly to the number of gateways stored at the trader, see figure 3. It takes longer to search for more complex gateways, and the search time get a stiffer slope as the number of gateways in the trader increases. The difference between local and remote search, and between searches which makes a match and those who misses, seems to be fractional. In the figure, the miss and match searches are joined together, and the average is presented.

## 5 Related Work

Supporting media processing within the network has been proposed as a solution for solving the heterogeneity problem. Usage of video gateways are suggested by Turletti and Bolot[14], as an alternative to layered coding, when the receivers have different bandwidth requirements. In [17] Yeadon et. al. creates a set of filters which implements and demonstrate the idea. Amir et. al. [1] design and implements a video gateway to address the problem of heterogeneity, in particular difference in bandwidth.

Work on locating media gateways have been an area of research. In the AGLP control protocol proposed by Ooi and van Renesse [10], programmable Internet servers which process multimedia stream are discovered. The protocol requires clients to upload the media processing gateway program. A multicast extension is proposed, in which several clients can take advantage of the same media gateway. The search criteria when locating a media gateway is its processing capabilities, and placement along the source to sink path. Placement is of interest in order to take advantage of bandwidth reduction as much as possible. The work of Xu et. al. [16] also deals with locating media gateways in the MeGaDip protocol. This protocol takes location and resource situation into account when discovering gateways.

The BMA protocol proposed by Rosenberg and Schulzrinne [8] deals with issues in locating gateways to the public switched telephone network, within the Internet. It is tailored toward voice over IP applications. The Service Location Protocol [15]

propose standards for locating services within the Internet.

Microsoft's DirectShow [2] and Sun's Java Media Framework [9] uses the concept of filters. They are components which can be plugged together to process media streams. They are mainly targeted used at the endpoints, as part of the application.

Our work differs from the above in that we emphasizes the description of media gateways. In particular we build our work on a type model for describing flow types. Gateways can be inserted anywhere in the distributed computing environment. Our work does not address the issue of resource availability or physical location of gateways.

## 6 Conclusions and Further Work

In this paper, usage of media gateways within a binding framework, have been investigated. In particular the requirements needed to automatically inserting gateways between participants in a binding, in order to overcome heterogeneity between the participants. The requirements for specifying gateways was investigated and a model was proposed. Our previous work on specifying interfaces was extended, for this purpose. Trading was proposed as a method to locate gateways, and a prototype based on the CORBA trader have been implemented and evaluated.

The use of the CORBA trader for media gateway trading, creates extra burden on the filter factory, which have to take care of shortcomings of CORBA trader. We plan to build a special purpose trader for trading media gateways, which will overcome these shortcomings.

In this work it is assumed that all gateway offers are equally good with regard to available resources and location. This will generally not be the case, and the gateway framework should be extended to take the current resource situation into account when considering gateway offers. Also the proposed framework is mostly tailored toward local area systems, where the location of the gateway does not matter that much. In further work, we plan to extend our gateway framework to take both resource and location considerations when trading gateways. The current work assumes that only a single gateway can be located to do the translation. If further work we plan to investigate the possibility of chaining multiple gateways, given that there can not be found a single gateway capable of doing the transcoding.

## References

- [1] Elan Amir, Steven McCanne, and Hui Zhang. An Application Level Video Gateway. In *ACM Multimedia 95*, pages 255–266, San Francisco, California, November 5-9 1995.
- [2] Microsoft Corporation. DirectX 8, DirectShow, 2000.
- [3] F. Eliassen. A Conformance Relationship for Stream Interfaces. In *2nd Int'l Conf on Formal Methods in Open Object-based Distributed Systems (FMOODS'97)*, Canterbury, July 1997.

- [4] F. Eliassen and S. Mehus. Type Checking Stream Flow Endpoints. In *Middleware'98*, pages 305–322, The Lake District, England, September 1998. Chapman & Hall.
- [5] F. Eliassen and J. R. Nicol. Supporting Interoperation of Continuous Media Objects. *Theory and Practice of Object System*, 2(2):95–117, 1996.
- [6] T. Fitzpatrick, G.S. Blair, G. Coulson, N. Davies, and P. Robin. Supporting Adaptive Multimedia Applications through Open Bindings. In *4th International Conference on Configurable Distributed Systems (ICCDs'98)*, Annapolis, Maryland, USA, May 1998.
- [7] Object Management Group. CORBA services: Common Object Service Specification. Technical report, Object Management Group, Framingham, MA, 1997.
- [8] J. Rosenberg and H. Schulzrinne. Internet Telephony Gateway Location. In *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, San Francisco, California, USA, March/April 1998.
- [9] Sun Microsystems. Java Media Framework, version 2.1.1, 2001.
- [10] Wei Tsang Ooi and Robbert van Renesse. An adaptive protocol for locating programmable media gateways. In *ACM Multimedia 2000*, Los Angeles, California, USA, October/November 2000.
- [11] Thomas Plageman, Frank Eliassen, Vera Goebel, Tom Kristensen, and Hans Ole Rafaelsen. Adaptive QoS Aware Binding of Persistent Multimedia Objects. In *Proceedings of International Symposium on Distributed Objects and Applications (DOA'99)*, Edinburgh, Scotland, September 1999.
- [12] H.O. Rafaelsen and F. Eliassen. Trading and Negotiating Stream Bindings. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware'2000)*, New York, USA, April 2000.
- [13] D. C. Schmidt, D. L. Levine, and S. Mungee. The Design and Performance of Real-Time Object Request Brokers. *Computer Communications*, 21:294–324, April 1998.
- [14] T. Turletti and J-C. Bolot. Issues with multicast video distribution in heterogeneous packet networks. In *Proc. 6th International Workshop on PACKET VIDEO*, pages F3.1–3.4, Portland, Oregon, 26-27 September 1994.
- [15] J. Veizades, E. Guttman, C. Perkins, and S. Kaplan. *Service Location Protocol*. IETF RFC-2165, June 1997.
- [16] Dongyan Xu, Klara Nahrstedt, and Duangdao Wichadakul. MeGaDiP: A Wide-Area Media Gateway Discovery Protocol. In *19th IEEE International Performance, Computing, and Communications Conference (IPCCC 2000)*, Phoenix, AZ, USA, February 2000.
- [17] Nicholas Yeadon, Francisco Garcia, David Hutchison, and Doug Shepherd. Filters: QoS Support Mechanisms for Mulipeer Communications. *IEEE Journal on Selected Areas in Computing (JSAC) special issue on Distributed Multimedia Systems and Technology*, 14(7):1245–1262, September 1996.