**Poseidon House**
**Castle Park**
**Cambridge CB3 0RD**
**United Kingdom**

TELEPHONE: **Cambridge (01223) 515010**
INTERNATIONAL: **+44 1223 515010**
FAX: **+44 1223 359779**
E-MAIL: **apm@ansa.co.uk**

**ANSA Phase III**

# CORBA and Streams: a White Paper

**Andre Kramer**

**Abstract**

This document is a position paper on CORBA and Streams. The aim of the proposal is to extend CORBA with the ODP concepts of streams and explicit binding. It is meant as input for the January OMG Telecommunications SIG.

APM.1684.00.01 **Draft** 5th January 1996

Standards Contribution

**Distribution:**
**Supersedes**:
**Superseded by**:

# CORBA and Streams: a White Paper


# CORBA and Streams: a White Paper

Andre Kramer

APM.1684.00.01

5th January 1996

The material in this Report has been developed as part of the ANSA Architecture for Open Distributed Systems. ANSA is a collaborative initiative, managed by Architecture Projects Management Limited on behalf of the companies sponsoring the ANSA Workprogramme.

The ANSA initiative is open to all companies and organisations. Further information on the ANSA Workprogramme, the material in this report, and on other reports can be obtained from the address below.

The authors acknowledge the help and assistance of their colleagues, in sponsoring companies and the ANSA team in Cambridge in the preparation of this report.

Architecture Projects Management Limited

Poseidon House
Castle Park
CAMBRIDGE
CB3 0RD
United Kingdom

| | |
|---|---|
| TELEPHONE UK | (01223) 515010 |
| INTERNATIONAL | +44 1223 515010 |
| FAX | +44 1223 359779 |
| E-MAIL | apm@ansa.co.uk |

# Contents

# 1 Introduction

Distributed multi-media systems require mechanism to implement and manage continuous flows or streams of multi-media data. Applications must be able to access these streams in a type safe manner, sending or receiving typed data frames, and must be able to control and manage streams via operational interfaces. It must also be possible to implement new kinds of streams, either to support new interaction/binding models such as broadcast conferencing, or to efficiently interface to new network transports or to novel hardware devices.

This note outlines a proposal for multi-media streams and explicit binding in OMG CORBA. We aim to support all the stream and binding capabilities defined by ODP. This proposal is kept as simple and as close to the 'spirit' of CORBA as possible: CORBA generally lacks some of the computational / type complexity of ODP (such as multiple interfaces per object and multiple terminations). We also try to maintaining backward compatibility with the CORBA standards as much as possible.

# 2 Format Types

Streams allow frames to flow asynchronously from a producer to one or more consumers. Frames contain application values or primitive media types and are typed: here these types are known as Format Types. Both producers and consumers use a Stream Interface to access a stream. A stream interface lists a set of format types (c.f. CORBA object interface and operations). Frames may be further grouped in named flows, adding an extra level of naming on frames.

One aim is to keep additions to the CORBA IDL to a minimum. In fact, it may be feasible to keep the IDL unchanged to maintain standard conformance. For example, one-way operations could be re-interpreted as stream frames.

Alternatively, stream interfaces could be viewed as CORBA operational interfaces extended with a list of one-way frames:

**Figure 2.1: Stream IDL**

```
interface stream_name {

     // normal operational part

} streams {

     frame1_name (arg1_type arg1_name); // declare frame.

     frame2_name (arg1_type arg1_name, arg2_type arg1_name);

};
```

The extended IDL declaration describes both the producer and consumer interface to the stream and is used to automatically generate the stubs for stream access. The interface's operational part typically declares control and management operations. Together these may be viewed as declaring a binding type. The server side for these binding operations is considered part of the stream infrastructure.

Marshalling the above stream, for example when passing the stream reference in an operation invocation (i.e. marshalling its interface reference), marshals the object reference for interface **`stream_name`** in the obvious manner (Note also that one can not name frames nor refer to CORBA operations remotely.)

Frames are like operations but with no return type (may map to void) and no argument directions: c.f. the CORBA default of INOUT and OUT or IN arguments. Frames also don't raise user defined exceptions.[1]

1. Type declarations in the streams interface extension are allowed but are viewed as textually part of the interface body.

# 3 Stream Directions and Named Flows

A stream interface has a set of **Frame Names** on which applications can both send and receive frames. As a refinement, directional attributes could be expressed via pragmas or more directly in the IDL:

**Figure 3.1: Stream Direction**

```
interface stream_name {

} streams {

     direction frame_name ();

     ...

};


direction ::= <none> | in | out | inout | cast?
```

The default being inout. Directions **in** or **out** limit the type of access possible to either consumer only or producer only respectively. Such frames could be used to interface to uni-directional hardware, or wrapped software that only allows one way flows. Stream stubs would then only be generated for the given direction. This facility also allows one to define separate IDL interfaces (possibly sharing common binding/management operations) for the sources and sinks of asymmetric streams if desired.

Optionally, the **cast** pragma can indicate potential one to many communication and may enable optimized stubs to be generated.

Inheritance may be used to compose stream types. CORBA does not support multiple interfaces to an object (relying only on inheritance mechanisms instead of multiple interfaces on an object). Stream interface composition via inheritance uses the normal CORBA rules.

A further refinement would allow multiple format types to be grouped into named **Flows** [1]. Flow names would be mediated by the stream implementation (frame ordering typically being preserved) and be reported to the consumers.

**Figure 3.2: Named Flows**

```
interface stream_name {

 // normal operational part

} streams {

     flow flow_name1 {

          frame1_name (arg1_type arg1_name); // declare frame.

          frame2_name (arg1_type arg1_name, arg2_type arg1_name

     }

     flow flow_name2 {

          frame1_name (arg1_type arg1_name); // declare again.

          frame3_name (arg3_type arg3_name);

     }

};
```

1. Nesting could be allowed.

# 4 Stream Binding

A stream interface must be bound locally before frames are sent on the stream. Binding connects the stream interface to the local stream infrastructure and may reserve stream resources. Typically the binding is established by invoking one of the normal operations of the interface.

Stream interfaces can be passed by reference in the usual manner (even when locally unbound). It would be possible to raise NotBound exceptions if a stream interface has not been explicitly bound locally at the recipient before first use of one of the frame operations for sending. The interface could, however, be implicitly bound (and remoted) on first use, using one of the listed profiles in the interface reference. If the lowest common factor is an RPC such as IIOP, then the frames could be bound to equivalent one-way operations as a default. An exception would also result if the interface reference did not include any common RPC address profiles.

When binding an interface, derived types can be allowed to be used in place of their base types. Additional frames (in base types) may be implicitly bound when first used and any sends would be mediated by normal CORBA remote invocation. Alternatively the additional frames may raise exceptions if used before they are bound.

## 4.1 Stream Adapters

The details of how stream interfaces are bound to streams are examined next:

Rather than adding stream binders to CORBA, the concept of Object Adapter could be overloaded to implement explicit binding. In mappings to languages with sophisticated type systems, adapters can directly reflect flow and stream types as well as format types.

A **`create`** operation is normally provided by an adapter (c.f. ANSA factories) which returns a stream interface reference. The adapter would typically register an object implementing the operational (binding) part of the returned interface reference with the ORB (may use the dynamic object activation mechanism of the BOA).

In order to be able to use a stream locally, the stream must then first be bound using a local Stream Adapter. An interface reference may be bound either as a stream producer or consumer or both under control of the Stream Adapter used. CORBA already has Object Adapters (OA) for the server side. In some sense, a CORBA adapter can be viewed as a binder for the server side of interfaces. Streaming interfaces would have adapter(s) for both the send (client) and receive (server) sides.

The existing CORBA adapters can be viewed as prototype binders. On the server side, this typically allows binding using an upcall mechanism (possibly

using a pool of threads and object activation (a type of persistence available in the basic object adapter BOA)). The CORBA client adapter is null (as CORBA only has implicit binding). Introducing a client side object adapter to support explicit binding is a main feature of this proposal.

Current and future adapters could also be augmented by using the CORBA context mechanism. Information such as flow name, or sender endpoint name or address could be passed as part of the invocation context, allowing more information than frame arguments to be passed over streams to consumers.

Special adapters can be implemented which implement specific resource management policies, multiple receive queues (e.g. one queue per named flow) or which interface directly to synchronous programming languages (Esterel). In C++, templates may be used to implement adapters parameterized on stream interface types.

Highly stylised adapters (e.g. for multi-party bindings) could be generated by compilers from declarative stream and flow specifications (as suggested by the DIMMA work at APM).

For example, QoS negotiation could be supported in a declarative fashion via this approach.

An adapter would typically provide additional language level objects (n.b. these are not extensions of CORBA IDL) to implement concepts such as Endpoints or Stream Objects for grouping multiple flows. An adapter could also introduce language level Binding Objects used for stream management and control .

Specific stream adapters could include: one for HTTP-NG type sessions; one for same-process FIFO streams; ones for linking in in-band stream processing and type conversion; ones which provide support for frame / flow synchronization. Specific adapters may be standardized (c.f. the BOA) and have standard IDL interfaces. Stream interfaces could inherit binding operations from a standard base binding type (expressed as an IDL interface type).

## 4.2 Primitive Adapters

Certain format types may be regarded as primitive in that they are generated and manipulated directly by hardware or are standardized media encodings (e.g. MPEG) or on-the-wire representations. Multi-media ORB Implementations would provide primitive adapters and IDL descriptions for their primitive media type flows (these IDL descriptions can be subject to standardization). Such primitive adapters could include bit pipes, ATM adapters or MBONE encapsulated encoding types (such as PCMA voice, Motion JPEG etc).

### 4.3 Adapter Composition

Higher/Application level stream adapters can be build from the standard lower level ones. These adapters would provide sinks for some stream interface type(s) and sources for another (or for several others). This implies that streams can be connected and implemented efficiently purely locally. Adapters can be commercialized in this scenario since they support a given portable IDL. The primitive adapters are, however, specific to an ORB offering.

# 5 Stream Stubs

Allowing multiple adapters requires stubs for streaming interfaces to be generic across different adapters. This can be readily achieved on the client side by using generic marshaling support, at the cost of indirections, as is achieved for example, in APM DIMMA. Indirection may also be used to support runtime stream binding.

The server side may require special adapter specific stubs to be generated (depending on the language being mapped to). We would aim to avoid this as much as possible, providing a standard simple upcall or receive downcall stub interface to languages such as C/C++. Adapters and stubs would then be typically provided in both an upcall (synchronous) and (downcall) asynchronous version for consumers. The upcall versions are restricted to one outstanding upcall at a time so that ordering is preserved (though applications can add their own concurrency)[1].

### 5.1 Interface References

The address profile part of object references can be used to carry information for stream interfaces. This allows, for example, purely local binders to be constructed by including stream protocol addresses as profiles. Avoiding other extensions allows interface references to remain compatible with GIOP.

1. ISIS uses a similar mechanism to preserve causality.

# 6 Stream QoS

Stream adapters should handle parameterization via QoS specifications. Both stream endpoint creation and interface binding must be parameterized with QoS specifications. Policy selections should also be supported on adapter creation.

It should also be possible to select invocation QoS (though we avoid making QoS part of the IDL or part of a language mapping by having “set-invocation-QoS” binder operations on adapters).

Stream management (including QoS management) is performed by standard operations supported by adapters. Issues such as transport level framing should also be addressed. How stream adapters are constructed to respect QoS constraints is beyond this proposal.

# 7 Summary

This note proposes limited changes to CORBA IDL to support asynchronous streams. It also overloads the concept of object adapter to provide stream binding and protocol support. Client adapters are also proposed in order to support explicit binding. Streams are interfaces with both a media flow part (which must be explicitly bound) and a conventional operation part for stream binding and management.

A set of standard interface media types and primitive adapters are proposed to enable portable multi-media applications. Adapters should be light-weight, allowing streams to be composed locally via linked adapters (possibly using a declarative or dataflow configuration language).

Much further work is required: we have not specified mappings to implementation languages or provided primitive media types or concrete stream adapters.

# 8 Related Material

## 8.1 IMA Multimedia Systems Services

The IMA's MSS proposal avoided IDL or ORB modifications but was not strongly typed and lacked standard means to introduce new primitive or application level streams which we hope to provide via stream adapters.

## 8.2 Pipes

The OSF DCE supports pipes which allow asynchronous streaming and interleaving on byte stream oriented transports. The proposal made here is more general, in that we aim to support more general interaction models with stream adapters. The Mercury project from MIT also supported asynchronous named pipes.

## 8.3 OMG Telecom SIG: Streams White Paper

Here we sketch initial answeres to the questions posed in the white paper [2]:

**3.1. Q** What is a stream?

**A** a) An object.

Extended IDL has a streams part for interfaces (set of frames) as well as an operational interface part (this set of operations typically implement binding and management). Frame operations are mediated by Stream Adapters.

**3.2. Q** What part of streams should be defined by the OMG?

**A** b) or possibly c).

The stream management object service can be viewed as a collection of standardized stream adapters. Stream stubs may also be subject to some standardization in order to work with diverse adapters.

**3.3. Q** How many sources and sinks?

**A** ALL) may be supported by the set of adapters.Composing streams also allows multiplexing, merging and in-band processing of data flows.

**3.4. Q** Unidirectional or Bidirectional or Multidirectional?

**A** ALL) as IDL default of inout (bidirectional) leads to generation of both producer and consumer stubs. Possible to restrict IDL to in or out, e.g. in order to interface to hardware. Cast may also be supported in IDL (possibly generating different send stubs) or may be a feature of the stream adapter.

NOTE: Must be able to locally compose streams using local adapters.

**3.5. Q** How is stream management modelled?

**A** c) Stream adapters create objects and implement the (binding) operational interface of the CORBA 'object'. This conventional operational interface implements stream management.

**3.6. Q** Where/when do you specify QoS?

**A** d) the (standard) Stream Adapters handle QoS management.

**3 .7. Q** Who specifies/negotiates QoS requirements?

**A** a) or b) Depending on the adapter, either or both parties can specify QoS and adapters can support negotiation.

**3.8. Q** QoS End user or format/protocol perception?

A Left up to the stream adapter. One adapter can use another with more primitive QoS in its implementation.

**3.9. Q** Synchronizing multiple flows?

**A** Can introduce generic (higher order) adapters which impose synchronization. May be possible to do generically in some language mappings and not in some others (or not be fully statically typed).

**3.10. Q** Streams in normal interfaces or on their own?

**A** We argue strongly for normal interfaces and for option c) (Reading streams as format types here).

**3.11. Q** Inheritance and Streams?

**A** a) addition of streams (frame types) and operations.

**3.12. Q** Many protocols? May stream types?

**A** There may be may binding protocols expressed as binding types (operation part of interfaces) and may stream types (the streams part of an interface) compose-able via inheritance. Adapters implement specific combinations.

**3.13. Q** Streams and IDL

**A**

For a) type of information flowing on the stream (the frame type)

For b) optional directional attributes. For c) QoS not in IDL.

For d) have operations to manage the stream (implemented by adapter).

**3.14. Q** “stream interface reference”?

**A** We propose one IDL description and one interface reference type for streams and objects. Implicit binding may be possible and would typically direct calls to the adapter which created the reference.

## 8.4 Example

Using a local adapter and a primitive adapter to implement a TV stream.

**Figure 8.1: TV Broadcast**

```
// A TV filter -- composite or segregated V/A to TV frames.
interface TV {
     bind_Rx(Composite_Broadcast composite_stream);
     bind_Rx(Separate_Video_Audio_Broadcast); // not used.
} streams {
     TV_frame(VideoFrameWithSound);
};

interface Broadcast { // models broadcast style transports
     bind_address(address, QoSparams);
};
interface Composite_Broadcast : Broadcast {
     bind_consumer(Composite_Broadcast);
} streams {
     video_frame(bytes);
     sound_bite(bytes);
}
interface VideoDevice {
} streams {
     ...
};
```

**Figure 8.2: Producer**

```
Composite_Broadcast_Ref ps =
          Composite_Broadcast_Adapter::create(QoS);

ps.bind(conference_addr, QoS); // set up send broadcast address
while (1) ps.video_frame(vd.getnextframe()); // send next frame
```

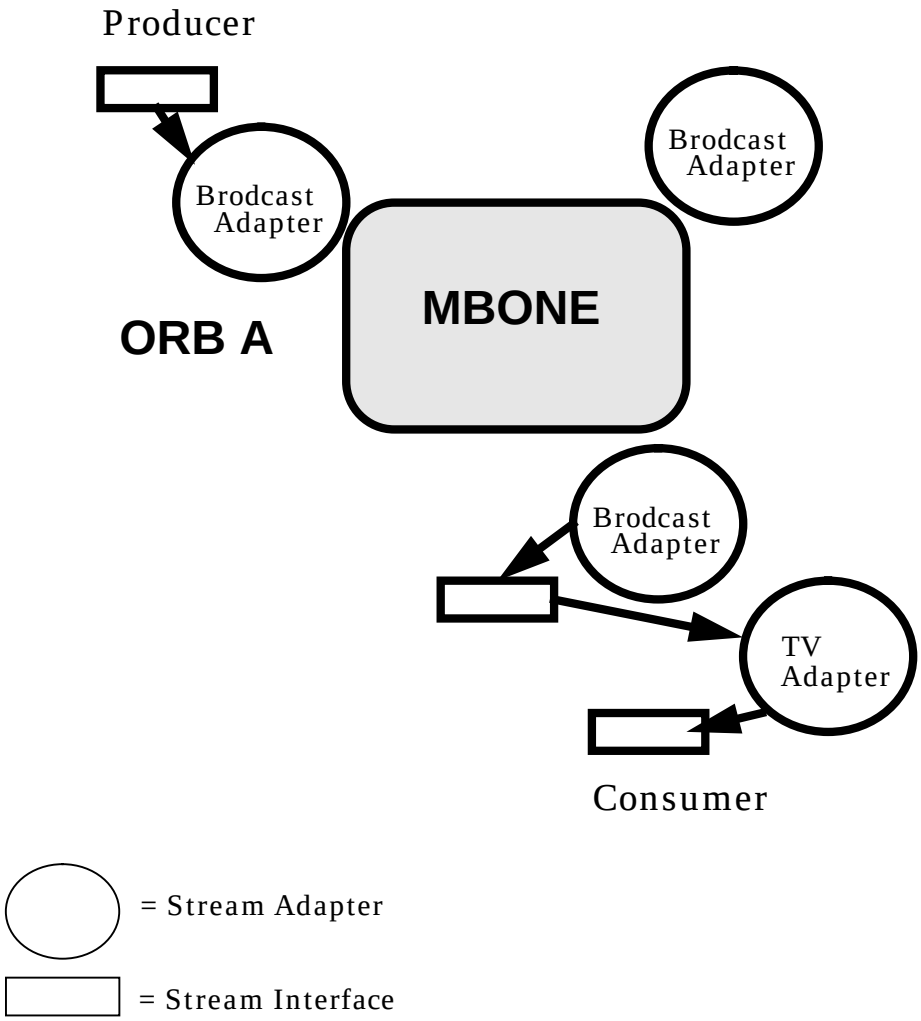
**Figure 8.3: Consumer**

```
// application's receive upcall for stream interface TV
//
void TV::recv_TV_frame(VideoFrameWithSound& vf) {
     play(vf);
}

Composite_Broadcast_Ref cs =
          Composite_Broadcast_Adapter::create(QoS);

// obtain ps stream reference from the producer.
cs.bind(ps); // set up recv broadcast address
TV_Ref tvs = TV::create();
tvs.bind_Rx(cs); // starts upcall processing.
```

**Figure 8.4: MBONE Application**



# References

[APM.1392]

The ANSA Binding Model, APM Ltd, Cambridge UK, Jan 1995.

[2]

Streams and QoS: a White Paper, Kerry Raymond,
OMG Telecommunications SIG, Dec 1995.