Native ATM support for CORBA Platforms

A. Puder and M. Moscarda
International Computer Science Institute
1947 Center St., Suite 600
Berkeley, CA
94704–1198
USA
{puder,moscarda}@icsi.berkeley.edu

Abstract

The CORBA standard as published by the OMG defines among other things the structure of a middleware platform called an *Object Request Broker* (ORB) in order to achieve interoperability of applications in distributed and heterogeneous environments. The standard does not prescribe a specific technology for building this middleware platform. In this paper we show how to integrate ATM technology into a CORBA compliant implementation. Making use of the advantages offered by ATM requires the modification of the ORB API. A prototype is based on the freely available CORBA implementation called MICO. We show how to make use of MICO's micro–kernel architecture in order to achieve a seamless integration of a new transport layer.

Keywords: CORBA, ATM Inter–ORB Protocol, micro–kernel ORB, special purpose object adapter.

1 Introduction

The *Common Object Request Broker Architecture* (CORBA) describes the architecture of a middleware platform which supports the portability and interoperability of applications in distributed and heterogeneous environments (see [3]). The CORBA standard is issued by the *Object Management Group* (OMG); an international organization with over 750 information software vendors, software developers and users. The goal of the OMG is the establishment of industry guidelines and object management specifications to provide a common framework for application development.

One important aspect of CORBA is that it is a *specification* and not an *implementation*. CORBA just provides a framework how applications can be distributed in a heterogeneous environment. But it does not prescribe any specific technology on how to implement the CORBA standard. We have implemented a freely available and fully compliant CORBA 2.0 implementation called MICO (the acronym recursively expands to **MICO Is CORBA**; see [4]). The architecture of MICO allows easy integration of new transport protocols without violating CORBA compliance. In this paper we show how to add native ATM (see [1]) support to the MICO architecture. The outline of the paper is as follows: in Section 2 we give a brief overview of GIOP and IIOP. In Section 3 we discuss the benefits of using ATM technology for a CORBA platform. In Section 4 we describe the design of the *QoS Inter–ORB Protocol* (QIOP) and the *ATM Inter–ORB Protocol* (AIOP). In Section 5 we provide some details on the prototypical implementation using the MICO ORB. In Section 6 we give an overview of the related work and in Section 7 we conclude our paper.

2 GIOP and IIOP

The CORBA standard defines an *Interface Definition Language* (IDL) which allows the specification of interfaces between components of an application. The *Object Request Broker* (ORB), a middleware platform defined by the CORBA standard, provides a communication layer by which those components of an application exchange information. An application programmer does not need to know how operation invocations are handled by the underlying ORB. We call the interface between application programs and an ORB the *horizontal interface* (see Figure 1).

The specification of the horizontal interface is sufficient as long as the application uses only one ORB.

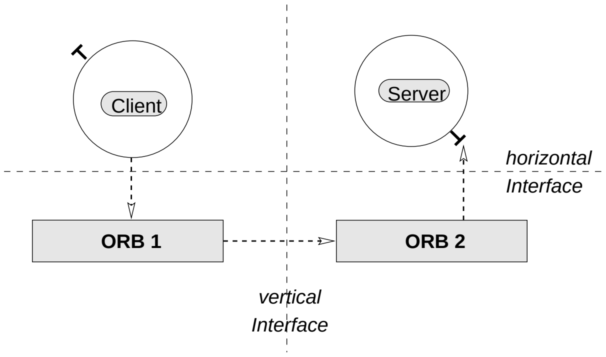

Figure 1: Horizontal and vertical interfaces in a CORBA environment.

But CORBA is only a specification and different ORB vendors may make different implementation decisions, e.g. for ORBs supporting security or fault–tolerance. One major advantage of the CORBA standard is that it also defines a *vertical interface*: the interface between two different ORBs. If every ORB vendor implements this interface, then it should be possible to distribute an application across different ORBs, exploiting specific features of each ORB implementation.

The vertical interface is mostly defined by the *General Inter–ORB Protocol* (GIOP) which describes the on–the–wire representations of basic and constructed IDL types as well as message formats needed for the protocol. The design of GIOP was driven by the goal to keep it simple, scalable and general. Furthermore the GIOP makes some assumptions about the underlying transport protocol. The main assumptions are that the transport layer is connection oriented, reliable and models a continuous byte stream.

The GIOP can be mapped to different technologies meeting those assumptions. One obvious candidate is TCP, which is a de–facto standard in the Internet. The *Internet Inter–ORB Protocol* (IIOP), which is part of the CORBA standard, shows how to connect different ORBs using TCP. It should be noted that IIOP is not a completely separate protocol from GIOP, but rather an instantiation of GIOP. In fact the transfer syntax and message formats defined in GIOP are also applicable for IIOP. The only elements added by IIOP are TCP specific things like network addresses or connection establishment.

3 Why CORBA and ATM

We designed a new protocol to integrate ATM in the CORBA platform and to use ATM connection for ORB interoperability. Several protocols could be used to implement the ORB interoperability, such as IPX or SNA, but ATM is of particular interest, because of the features ATM offers and the goals that object–oriented middleware platforms — such as OMG CORBA — try to accomplish.

For example, there is an increasing demand to extend object–oriented middleware to support applications with stringent *Quality of Service* (QoS) requirements. However, conventional CORBA implementations incur high latency and low scalability when used for performance–sensitive applications. Due to the excessive marshaling/demarshaling overhead, data copying and high levels of function call overhead, conventional implementations of IIOP yield poor performance over high speed networks. These inefficiencies discourage developers from using CORBA for mission/life–critical applications. Furthermore, when CORBA will support stream interfaces in a future revision, transport protocols like TCP will not be suitable for supporting critical real–time or isochronous applications due to their higher processing overhead. At the present time there are several studies on how optimize IIOP performance over high speed network.

ATM is particularly attractive mainly due to its low cost, scalable performance and its potential support of geographically distributed computing. Furthermore, native ATM allows software designers to create applications without TCP/IP protocol stack and the socket layer buffering mechanism. Our approach provides object–oriented application programming interfaces to facilitate rapid development of applications involving the transmission of continous media over ATM.

There are a lot of possible network technologies, but ATM emerges as the most suitable to support future applications and high speed communications for both local and wide–area networks. We have chosen to use the ATM Adaptation Layer 5 (AAL5), which provides connection oriented, but unreliable end–to–end connections. This could be the first step to provide a stream service for continous media applications with end–to–end QoS guaranteed while retaining the merits of CORBA.

4 QIOP and AIOP

Since GIOP is designed to work directly over any connection–oriented transport protocol that meets a minimal set of assumptions, we could choose ATM as a possible technology to support ORB interoperability. The problem is that ATM is not the most suitable protocol to implement GIOP, because, unlike TCP, ATM doesn't meet all the assumptions that GIOP makes on the underlying transport protocol. According to the main GIOP assumptions, the transport protocol must be connection oriented and reliable. AAL5 does not provide a reliable delivery mechanism; it is up to end systems to provide end–to–end reliability where it is needed.

However, neither GIOP nor IIOP provide any detailed support for specifying or enforcing the end–to–end QoS requirements of the application. This makes GIOP/IIOP unsuitable for real–time applications that cannot tolerate the latency overhead and jitter of TCP/IP transport protocol. TCP functionality, like adaptive retransmissions, deferred transmissions and delayed acknowledgments can cause excessive overhead and latency for real–time applications.

Since ATM could be useful to implement ORB interoperability, we have designed a new protocol, called *QoS Inter–ORB Protocol* (QIOP), to enhance performance and to allow the ORBs to communicate through ATM. By simply modifying the GIOP assumptions, QIOP allows CORBA to use ATM to implement the interoperability between ORBs. In particular we refer to the mapping of QIOP onto the ATM connections as the *ATM Inter–ORB Protocol* (AIOP).

4.1 Framework for ORB interoperability

The basic idea in our work is to modify the GIOP assumptions to make ATM the most suitable technology for ORB interoperability. Although by removing the assumption about reliability, the CORBA framework itself puts some contraints about reliability. These contraints are formulated in terms of failure semantics of operations which must be met by any CORBA compliant implementation. During the design of QIOP we will need to make sure, that these constraints are not neglected.

More in detail, two types of failure semantics are defined by the CORBA specification:

At–most–once: if an operation request returns successfully, it was performed exactly once; if it returns an exception indication, it was performed at–most–once.

Best–effort: a best–effort operation is a request–only operation (indicated by the `oneway` keyword in CORBA–IDL). With this failure semantics no guarantee whatsoever is given concerning the execution of an operation.

These failure semantics for operations implicitly assume a certain kind of reliability which must be supported by any transport layer used for the transmission of operations. These implications are not visible in GIOP, since GIOP itself requires reliability for the transport protocol. But in the case of QIOP, our Inter–ORB protocol described in the next section, the CORBA assumptions have a stronger impact, since we will drop reliability requirements for QIOP.

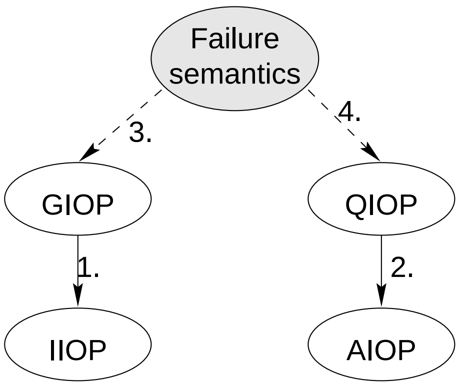

Figure 2: Dependencies between different transport protocols and the influence of the failure semantics.

By removing the reliability as a requirement for the transport, we make ATM the most natural technology to implement ORB interoperability, but on the other hand we cannot simply drop this assumption without any other modification, because the failure semantics have an influence on the QIOP, as shown in Figure 2.

Just as IIOP is an instantiation of GIOP (arrow 1 in Figure 2), AIOP is an instantiation of QIOP (arrow 2). The failure semantics for operation invocations that are prescribed by the CORBA standard, have influence on the design of GIOP and QIOP (arrows 3 and 4). This is particularily true for QIOP, since is does not assume a reliable transport layer. In order to ensure the failure semantics, some kind of reliability mechanism has to be introduced to QIOP. We will describe this in detail in the next section.

4.2 Design of QIOP

Since it is not possible to take advantages of the ATM benefits using GIOP, we designed a new protocol, called *QoS Inter–ORB Protocol* (QIOP), to allow the usage of ATM connections in a CORBA environment. QIOP is a specific version of GIOP which is more suitable to be mapped onto ATM. It should be noted that the modification of the vertical interface also induces modifications of the horizontal interface, otherwise the application could not take advantage of the features of QIOP. The extended API between the

ORB and an application will be discussed in one of the following sections. The following objectives were pursued with the design of QIOP:

simplicity: we want QIOP to be as simple as possible. This is the main reason for modifying GIOP to allow a direct mapping onto ATM connections.

generality: as GIOP, QIOP can also be mapped onto other connection–oriented transport protocols that meet a minimal set of assumptions.

QoS support: this is the main target we wanted to achieve and this is an objective that is not considered in the GIOP design; QIOP must be simple, but at the same time it must be able to support QoS requirements of an application.

The specification of GIOP consists essentially of the GIOP message formats and the GIOP transport assumptions. GIOP cannot be mapped directly onto ATM connections mainly due to the problem of reliability. The QIOP definition makes the following new assumptions regarding transport behavior:

- The transport layer is connection–oriented: as GIOP, QIOP uses connections to define the scope and extent of request IDs.

- The transport layer supports and allows to guarantee some QoS parameters: in particular for the ATM mapping it is possible to specify QoS parameters such as peak bandwidth, mean bandwidth and the average size of packets sent at peak bandwidth.

- The transport can be viewed as a byte stream and no arbitrary message size limitations are enforced.

With these new assumptions, the protocol must support QoS requirements and must allow a resource reservation for the end–system. In this case, TCP is not the most suitable protocol to implement ORB interoperability anymore. As a matter of fact, TCP doesn't meet the QIOP assumptions, but ATM does. Now it is possible to implement the vertical interface using ATM features to specify QoS parameters, such as bandwidth for a given connection.

Although we do not assume a reliable transport layer in order to facilitate the usage of ATM technology, we still need to assure the failure semantics as prescribed by the CORBA standard. As mentioned before, the failure semantics are at–most–once for normal operations and best–effort for oneway operations. In order to meet these requirements, the QIOP must

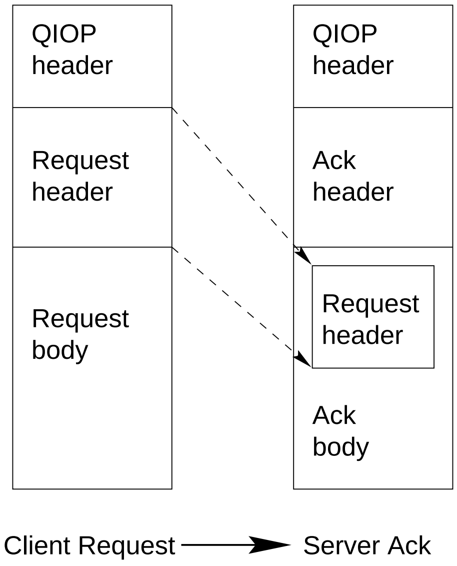

Figure 3: The request header is encapsulated in the `Ack` body.

provide means for explicit support of these failure semantics. Note that this is not necessary for GIOP since it already assumes a reliable transport layer.

For this reason we decided to modify the specification of the vertical interface by adding to the eight message types of GIOP version 1.1 another message type, in order to ensure the failure semantics. We refer to this new message as `Ack` and by using this kind of acknowledgment both the client and the server can specify the successful delivery of a request or a reply.

`Ack` messages can flow from client to server or vice versa and have three elements, encoded in this order:

- a QIOP message Header

- a `Ack` Header

- the `Ack` Body

The `Ack` message is sent following a previous `Request` or `Reply` message if an appropriate bit in the QIOP header is set to TRUE in the flags field.

For example assume that a client wants to send a `Request` message to a server. If the application requires an acknowledgment to confirm the receiving of a request, it sends the `Ack` message back by encapsulating the request header in its body, as shown in Figure 3.

The last point we have to emphasize concerns the impact of these modifications to the horizontal interface. We modified the GIOP in order to build a new protocol and we added new features to the vertical interface. However, there is a strong correlation between the horizontal and the vertical interface in a CORBA environement and they are not independent from each other. Without any modification to the horizontal interface an application could not make use of the new QoS features. Therefore the API between the ORB and an application needs to be extended.

It is important to note that for the scope of this paper we are only concerned about network specific QoS, not application specific QoS. The latter would require a modification to the IDL language since application specific QoS parameters should be specified at this level. Since we are only interested in network specific QoS as defined by the ATM technology, it is sufficient to modify the ORB API. This will be explained in one of the following sections.

4.3 Design of AIOP

ATM can serve as a possible transport layer for QIOP. We have used ATM APIs libraries for mapping QIOP message transfer to ATM connections and we refer to this specific mapping as the *ATM Inter–ORB Protocol* (AIOP). Just as IIOP can be seen as an instantiation of GIOP, AIOP is an instantiation of QIOP. Whereas IIOP uses TCP/IP as a transport layer for GIOP messages, AIOP uses ATM as a transport layer for QIOP messages.

We have chosen to use the ForeThought API from ForeSystems for the implementation of the prototype. This ATM library provides a connection–oriented client/server model. Before data can be transferred, a connection has to be established between client and server. Once a connection is established, data can be transmitted from one ATM application to another by using the ATM API.

Resources (bandwidth) are reserved for each connection. The most interesting feature is that this ATM library uses an appropriate structure to specify the bandwidth requirements; in particular it allows to specify the following parameters: the peak bandwidth, the maximum rate at which the source produces data, measured in kilobits per second; the mean bandwidth, the average bandwidth expected over the lifetime of the connection, also measured in kilobits per second; and the mean burst length, the average size in kilobits of packets sent at peak bandwidth. The network uses the bandwidth information in the connection request to reduce chances of buffer overflow as well as refusal of a connection if insufficient resources are available.

A server object is addressed through an *Interoperable Object Reference* (IOR), which contains a ATM address. Servers must listen for connection requests. A client requesting an object's service must initiate a connection with the address specified in the IOR with a connect request. The listening server may accept or reject the connection. Once a connection is accepted the client as well as the server may send messages by writing to the ATM connection. After sending (or receiving) a `CloseConnection` message, both client and server must close the ATM connection.

We can define AIOP profiles to identify individual objects accessible through the AIOP. AIOP profiles are referred to the server and have the following form:

```
module AIOP {
  struct Version {
    octet  major;
    octet  minor;
  };

  struct ProfileBody {
    Version                          aiop_version;
    sequence<octet,8>                atm_address;
    unsigned short                   sap;
    sequence<octet>                  object_key;
    sequence<IOP::TaggedComponent>   components;
  };
};
```

The QoS constraints are given by the client. Since the server is not involved in QoS constraints decision, it is not necessary to specify QoS parameters in the AIOP module. AIOP profiles specify the version of AIOP that the agent of the specific address is prepared to receive.

Furthermore, AIOP profiles identify the ATM host to which QIOP messages for the specified object may be sent and the connection used for requests. An ATM host consists of an ATM name, the hardware address of the target device, and a SAP (Service Access Point), that allows the call notification to reach the correct software component within that target device. Finally we can specify a value in the AIOP profile used in request messages to identify the object to which the request is directed and a sequence which contains additional information that may be used in making invocations on the object described by this profile.

5 Implementation of AIOP

In this section we outline the implementation of AIOP as described in the previous section. The implementation is based on the freely available CORBA implementation called Mico and the ForeRunner ATM

API from ForeSystems. As pointed out before, the CORBA specification defines a horizontal as well as a vertical interface in open distributed systems. The AIOP specifies the vertical interface between two ORB implementations. The enhanced functionality in terms of QoS has influence to the horizontal interface as well. In Section 5.1 we will first describe the implementation of the vertical interface and then in Section 5.2 the implementation of the horizontal interface.

5.1 Vertical interface

In the following we describe how we integrate the new ATM transport layer into our MICO implementation. The key issue of integrating a new transport protocol into the ORB relies on specific aspects of the CORBA specification. Before we explain our approach, we first review the tasks of an *Object Request Broker* (ORB) and an *Object Adaptor* (OA). According to the CORBA standard, the ORB is responsible for the following tasks:

- locate appropriate implementation
- transmit parameters & control to implementation
- return results to client

An *Object Adaptor* (OA), which is also described in the CORBA standard, provides the following services to an object implementation:

- handling of references
- implementation activation
- object creation

It should be noted that there can exist more than one OA. CORBA implementors can create specific OAs, which are tailored to some specific needs. E.g., a particular OA might support persistance or fault tolerance. The CORBA standard prescribes one OA, called the *Basic Object Adaptor* (BOA), which compliant implementations have to provide.

The difference between MICO and other CORBA implementations is, that we followed a micro–kernel approach: we moved as much functionality out of the ORB into dedicated OAs in order to achieve a modular and extensible implementation[1]. We even use this approach for CORBA specific things like support for different activation modes or implementation of IIOP. The following outlines the path of a method invocation from client to server to explain the micro–kernel architecture.

A method invoked by a client first enters the ORB. The ORB then selects an approriate OA to dispatch the operation to the target object. In the MICO architecture the dispatching of operations crossing different address spaces is also handled by special OAs (see Figure 4). The operation invoked by the client enters the ORB just as in the local case. This time the ORB forwards the operation to a special OA called the *IIOP Client*. This particular OA implements the client side of GIOP over TCP/IP. The *IIOP Client OA* sends the operation over the network to an object implementing the server side of GIOP over TCP/IP. This object (called *IIOP Server* in Figure 4) sends the operation to the remote ORB. The remote ORB finally dispatches the operation via its BOA to the target object.

In the MICO architecture the ORB has no knowledge about different transport media. These are handled by special OAs and special objects. Adding a new protocol is just a matter of introducing a new pair OA/object which register themselfes with the ORB. We integrated native ATM support into the MICO architecture as depicted in Figure 4. The client side of AIOP is implemented by a *AIOP Client*. It establishes a connection over native ATM to a corresponding *AIOP Server* object implementing the server side of GIOP.

5.2 Horizontal interface

As pointed out before, the horozontal interface in an open distributed system resides between the middleware platform and the application. If CORBA technology is used as a middleware platform, the horizontal interface constists of an API offering specific services to an application as well as interfaces defined through the IDL. All changes made to the vertical interface induce changes in the horizontal interface, otherwise the richer functionality of the vertical interface could not be used by the application.

Since IDL is ultimately used for inter–application interfaces (i.e. client/server interfaces) and AIOP only addresses network specific QoS offered by ATM, there is no need to make changes to the IDL. This would only be required if application–specific QoS parameters were included in the framework. It therefore suffices to make changes to the ORB API to give applications access to the QoS enhancements of the vertical interface.

[1] It should be noted that the CORBA standard does not prescribe a specific interface between an ORB and its OAs. The approach described here is therefore MICO specific and not portable accross different ORBs.

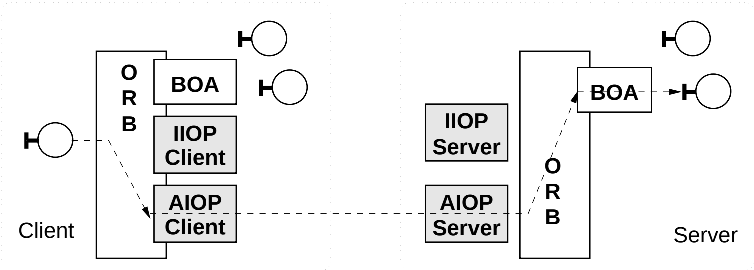

Figure 4: Special Object Adaptors (OA) provide access to different transport media.

Since ORBs may be written in different programming languages, the CORBA standard uses its own IDL to specify the ORB API in a language independant way. Although the API is specified through the CORBA–IDL, the ORB is not an object in the normal sense (e.g. there exist no stubs or skeletons for the ORB). To highlight this difference, the CORBA standard refers to a *Pseudo IDL* (PIDL) when defining the ORB API. In order to incorporate QoS capabilities in the horizontal interface, one must augment the PIDL specification of the ORB API.

In our implementation, the PIDL specification of the QoS specific portion of the ORB API looks as follows:

```
// PIDL
interface ORB {
  // Other operations offered by the ORB

  struct QoS_Value {
    unsigned short target;
    unsigned short minimum;
  };

  struct QoS_Param {
    QoS_Value peak_bandwidth;
    QoS_Value mean_bandwidth;
    QoS_Value mean_burst;
  };

  attribute QoS_Param qos;
};
```

ATM allows the target value (i.e. preferred value) and the minimum value of a QoS parameter. These are captured by the definition of `QoS_Value`. Those parameters apply to the peak and mean bandwidth and the mean burst which are described by the type `QoS_Param`. All QoS parameters are accessible through an attribute called `qos` which allows read and write access to those parameters.

It should be noted that those settings are per ORB, not per object. The idea is, that if a connection to a remote object has to be established, the current QoS parameter as specified through `qos` are relevant. If two remote objects which are managed by the same remote ORB are referenced, then only those QoS settings are relevant for the object referenced first by the client. If the QoS settings should change for the reference to the second remote object, then these will be ignored.

Applying the C++ mapping to the PIDL above, setting the QoS parameters in C++ can be done in the following way:

```
// C++
QoS_Param params;
// Initialize QoS parameters
params.peak_bandwidth.minimum = 2;
// ...

// Activate QoS parameters
orb->qos( params );
```

6 Related work

We designed QIOP and AIOP in order to integrate ATM technology into the CORBA architecture. Our main accomplishment is the possibility of adding ATM based QoS parameters to CORBA; in particular by using a suitable ATM API we can specify the bandwidth requested for a given connection. We presented a possible solution to provide QoS guarantees for real–time CORBA application, but there are several studies on how solve this problem.

The most important research related to this kind of work is currently being carried out at the Washington University in St. Louis. The Distributed Object Computing group at Washington University is developing a high–performance, real–time ORB called TAO (The ACE ORB; see [5]). The goal of TAO is to provide end–to–end quality of service guarantees by vertically integrating CORBA middleware with OS I/O subsystems, communication protocols and network interfaces.

The TAO motivations are the same we have: CORBA is not yet suitable for performance–sensitive, real–time applications due to the lack of QoS specification interfaces and enforcement. In our design we decided to build a new protocol to take advantage of ATM to specify the bandwidth and we utilized native ATM to implement ORBs interoperability. TAO introduces a *Real–time Inter-ORB Protocol* (RIOP) which represents another mapping of GIOP that allows applications to transfer their QoS parameters from clients to server objects.

When a message is sent from a client to a server, RIOP packages attributes, such as priority and execution period, in the message in order to specify QoS requirements. RIOP extends GIOP/IIOP with QoS attributes and in addition to transporting application QoS parameters, TAO's RIOP is designed to map GIOP on a variety of networks (including high–speed network like ATM/IP WANs).

The OMG has also issued a request for proposals (RFP) under the topic of real–time CORBA (see [2]). Several submissions have been made; each of them introducing a new protocol in order to support real–time features. One submission suggests to extend the CORBA framework by pluggable protocols in order to support different kinds of inter ORB protocols. As we have shown, the micro–kernel architecture is more flexible and allows the integration of functionality other than new protocols. We therefore propose to standardize the interface between the ORB and an OA, and not limit the scope only to pluggable protocols.

7 Conclusions

The CORBA standard defines a specification for the implementation of a middleware platform. It is mainly concerned with the specification of interfaces between different components. We have identified two different kinds of interfaces: a horizontal interface between the ORB and an application and a vertical interface between two ORBs. This general scheme allows the definition of new protocols without affecting other components of the middleware platform. As a proof of concept we have shown how ATM technology can be used to add network based QoS support in a CORBA platform while still maintaining CORBA compliance. For the implementation we have exploited the micro–kernel architecture of a freely available CORBA implementation called Mico. Future work will include application specific QoS which requires modifications to the inter ORB protocol as well as the IDL.

Acknolwledgements

The authors wish to thank Andreas Schultz for supporting our work. He implemented the prototype of AIOP based on Mico.

References

[1] The ATM Forum, http://www.atmforum.com.

[2] OMG. Realtime CORBA 1.0 RFP, http://www.omg.org/library/schedule/Realtime_CORBA_1.0_RFP.htm. TC work in progress, Object Management Group, 1997.

[3] Object Management Group (OMG), The Common Object Request Broker: Architecure and Specification, Revision 2.0, July 1995.

[4] A. Puder and K. Römer. MICO — a free CORBA 2.0
implementation. http://www.vsb.informatik.uni–frankfurt.de/~mico, Computer Science Department, University of Frankfurt, 1997.

[5] D.C. Schmidt, D.L. Levine, and S. Mungee. The Design and Performance of the TAO Real–Time Object Request Broker. In *Computer Communications*, 1997.