2

Quality of Service questions of stream objects built on CORBA

J. Jormakka
Helsinki Telephone Company, Research
P.O.Box 138, 00381 Helsinki, Finland,
E-mail:jorma.jormakka@hpy.fi
tel:+358 9 6064721, fax:+358 9 6064839

Abstract

Helsinki Telephone Company Research is currently involved in a project evaluating suitability of CORBA as a presentation layer distribution technology for streams.

One application of CORBA is connection management of streams, that is basically as a replacement of signalling for communicating objects transferring voice and video, for instance for video-on-demand or videotelephony. Another application is service management, like ordering videotelephony or video-on-demand service.

These kind of scenarios can be build using the ideas from TINA-C so that the operational and stream interfaces of TINA DCE are realised by CORBA and the stream transport by IP. The role of TINA is to clarify the relations of customers, retailers and third party content providers by mapping the relations to TINA reference points.

Quality aspects of the streams in this kind of solution are essential from many points of view. Firstly, a solution replacing signalling for streams must meet sufficient delay and blocking requirements similar to those of SS7. Secondly, passing QoS requirements to the underlying GIOP has to be investigated. Thirdly, reliability/robustness issues are very relevant in a transparent distribution method such as CORBA - how does the solution react to unavailability of a remote object and so on.

This paper does not present ready solutions but explains initial ideas and problematic.

Keywords

CORBA, QoS, GOS, Performance.

This paper is based on work done in the EURESCOM project "EURESCOM Services Platform". The project has the following participants: KPN Research , Finnet-group, British Telecom, Deutch Telecom, France Telecom and Telecom Ireland. The views presented are author's personal opinions.

Intelligent Networks and Intelligence in Networks D. Gaiti (Ed.)
Published by Chapman & Hall

1. WHY CORBA, WHY DISTRIBUTION ON PRESENTATION LAYER?

Loosely speaking a stream is a connection which carries video or voice and consequently has some real time performance requirements. One of the reasons why operators are interested in CORBA is that the choice of a network for carrying streams is not easy. Granted that the solution is a broadband network there are several alternatives:
- native ATM end-to-end: ATM with UNI 3.1/Q.2931 or UNI 4.0
- IP over ATM: some form of IP switching, MARS for multicast?
- Internet with QoS, implying RSVP, maybe also IPv6, NHRP?
- IP directly on SDH ? (though SDH hardly is sufficiently flexible for end-to-end connections)
- some other solution.

When the future solution or solutions are not known it is natural to separate application software from the network by a common interface situated somewhere between the layers 4 and the lower part of 7 in the OSI model. CORBA is one serious alternative. It corresponds rather well to the OSI presentation layer and the COSS services correspond to certain extent to the Common Application Service Elements of OSI. Many of the arguments in favour of CORBA as an improvement to the TCP/UDP socket interface are equally valid for the OSI presentation and belong to the normal reasons for using a presentation layer. While these are good arguments one must remember that OSI presentation layer did not become popular, therefore it is important to think what were the experiences learned from the OSI presentation when considering CORBA.

One of the reasons for the failure of the OSI presentation's popularity was that the OSI applications did not gain acceptance which depended on the difficulty of interworking and on the high cost, vendors also seemed to prefer to develop either proprietary solutions or solutions based on de-facto standards. CORBA has better chances since it is not a competitor but complementary to Internet protocols and relates to application distribution as a part of software development procedure rather than to multi-vendor interworking which is mainly in the interest of operators.

On the lower layers of OSI protocols there were problems in addresses. In many implementations simply setting a selected mode of X.25 addressing was too difficult for the user. Furthermore, X.25 was seen expensive and often OSI implementors decided also to support TCP/IP sockets as an alternative to X.25. With CORBA using IIOP should not have these problems but what about other GIOPs ?

Setting correctly the presentation address (NSAP, TSAP, SSAP, PSAP) in OSI implementations often took some time in interworking trials. Furthermore, in some OSI implementations optional parameters of ACSE, such as AE Titles and Access Controls were used as mandatory parameters in the meaning that interworking was not possible unless these parameters were used - causing interworking problems with implementations which lacked them. In CORBA these addressing related issues include GIOP addresses and representation of interoperable object references (IOR) and do not yet seem to contain major interworking problems (which is odd since IORs look both long and strange and they change often).

In OSI protocols there were many options which the implementations only partially supported. Even though functional profiles were created for agreeing on the options they remained an interworking problem. With CORBA it is possible that partial support of COSS services is a similar future problem.

The general impression given by OSI implementations was that the software was transferring data from one layer / one data structure to another too many times. Raw data could be passed as pointers but still the result was a large and possibly ineffective software. This problem appeared especially when considering the OSI presentation layer. It can be seen as a logical problem: coding and decoding from ASN.1 does not always fit in the 6th layer but partially could be done better on the 7th layer. When decoding from transfer syntax the presentation layer can only create some structures and decode data from the transfer syntax to these structures since it has no knowledge what to do with the application data. Later these structures usually are changed to other internal structures suitable for the application software. For instance in a CMISE implementations using XOM/XMP data from GDMO-defined MIBs could be taken to some internal structures which then are converted to the (rather clumsy) data structures passed to XOM/XMP and then the data is finally put to the transfer syntax. For performance and memory reasons it can be more tempting to code partially the transfer syntax in the application rather than to pass the data in internal structures to the presentation layer for coding to the transfer syntax, some X.500 implementations did so.

The above mentioned situation with OSI implementations is basically the same with CORBA but since there are less protocol layers the problem should be less apparent. Application development with CORBA does not look like if it would result to this large number of conversions between data structures typical to OSI applications.

In OSI presentation ASN.1 with BER coding rules is the transfer syntax and has been seen as a performance bottleneck. In CORBA IDL is closer to C++ and coding can be faster in principle. Still it is good to notice that the problem was not ASN.1 which is a good language, nor BER, but the complicated structures specified for the applications

that caused the performance problems in OSI, with IDL performance also depends on the structures used.

The design procedure in OSI applications is made by ITU and ISO standardisation and is based on state automatons, message sequence charts, description languages ASN.1, ASN macros, GDMO. CORBA based applications may use the OMG proposal of Object Oriented Analysis and Design using the Unified Modelling Language (UML). Concerning the design procedure, both methods produce an application that can (?) be implemented. It is difficult to say if implementation of CORBA based applications is easier than implementation of OSI applications. The comparison is difficult since thinking of OSI the writer has in mind some past development of large OSI applications following complicated standards whereas with CORBA implementation procedure only can think of some simple examples. Some comparison can be made between OSI management application implementation procedure and implementing a distributed application using CORBA, both being object based approaches - in fact, OSI TMN may change CMISE to CORBA in near future.

A development environment for OSI management applications can for instance include compilers for ASN.1 and GDMO, a ready OSI-stack up to TLI, session, presentation or even application - one possibility being the XOM/XMP interface. A user writes GDMO and ASN.1 definitions and compiles in principle automatically the MIBs. In practise a programmer most probably has to have knowledge of the structures and functions produced. This procedure in OSI management is untypical for development of OSI software which is structured around implementations of service elements as automatons. Object based implementation procedures with OSI are therefore connected with object based applications (CMISE, X.500) and there is no general procedure of this kind.

A CORBA application is usually developed with the IDL approach. Alternative ways are DII for the client and DSI for the server. There is also ORB interface for client and server but it has only some operations. Implementation of CORBA determines the actual development procedures, in IONA's Orbix the CORBA software basically consists of client and server libraries and orbixd daemon. In the IDL approach IDL definitions are first written and the IDL is compiled to the chosen programming language (e.g., C++). This compilation produces for a client an IDL stub and for a server object an IDL skeleton. The server object can inherit from a Base Object Adapter, BOA Approach, or is made with the TIE Approach. The server is registered to the ImplementationRepository and then can be used by clients. The daemon orbixd reads the ImplementationRepository database to see which object implementation is activated.

A main difference in CORBA and OSI is the philosophy of distribution. OSI applications are built on the concepts of an agent and a server and the distribution is on

the application layer. CORBA makes the distribution more transparently using stubs. The method used by CORBA is occasionally given as a major improvement but many styles of distribution have their own advantages. Other distribution methods include calling interfaces like the TCP/UDP socket interface or TLI, transparent distribution of files by remote calls being made by the application invoking normal local operating system calls if a file is remote (like in NFS). Finally there is distribution made on the user interface level like in X. These are all good methods of distribution but object-oriented distribution on presentation layer is currently more interesting than the other solutions because of the strong support to distributed processing from ISO/ODP, TINA and naturally OMG.

Another difference in CORBA is emphasis on stateless operation. However, there are applications which can gain from automatons and very possibly all future applications based on CORBA will not be stateless.

As a conclusion, distributing applications using a presentation layer is advantageous especially since there are many network alternatives. Distribution using object-oriented approach can apply ideas from ISO/ODP and TINA and CORBA is a practical way for implementing those ideas, yet OMG CORBA has MicroSoft OLE as a strong competitor. However, concerning multi-vendor interoperability problems CORBA uses similar ideas as the OSI presentation and the (inter)working problems appeared with applications.

Because of the transparent way of distribution performance and reliability aspects in CORBA are more important than in non-transparent distribution methods. Four questions can be posed:

- What is the impact of the CORBA layer between application and GIOP to delays?
- How to pass QoS parameters from application to stream transmission protocol through CORBA?
- What are the reliability/robustness issues?
- What about traffic/congestion controls?

Since streams and streams connection management are the interesting applications of CORBA in this paper, the discussion will be limited to such usage.

2. APPLICATION OF CORBA TO VOICE AND VIDEOTELEPHONY

CORBA suits well to cases when a large application should be distributed transparently. It seems unnecessarily complicated for basic communication tasks like sending voice or video from one place to another, why not simply use RSVP for signalling and IP for streams. However, applications tend to become more complicated in time and there is

need for something like CORBA. In the project described here the goal is to gain some experience on CORBA with streams and the simple application must be seen in this light, it is only a simple case which could be done easier but we are interested in applying CORBA.

One possible solution using CORBA with streams is to implement videotelephony directly on an ATM network using CORBA for signalling. A solution of this kind is xbind [10] which calls directly UNI. Since the future popularity of native ATM solutions is a question mark, this type of solution was not selected.

Another possibility is that signalling goes through CORBA and IP carries streams. Unspecified quality using IPv4 is probably not sufficient in future but a suitable QoS can be reserved with RSVP and the streams carried by IPv6. This alternative is also being investigated but in another project.

The selected solution is loosely based on TINA. In the TINA concept CORBA plays the role of KTN (Kernel Transport Network). TINA-C has defined stream interfaces and one possibility was to interpret this so that the streams would actually go through CORBA. Knowing that present implementations of CORBA actually bring a considerable overhead to streams over simple socket based transmission, it is a more practical solution to use directly Internet protocols for the streams omitting CORBA even though the concepts are expressed as far as possible in TINA-C terminology, so in this way the solution is only loosely TINA. Connection management (signalling) is made through CORBA and corresponds to TINA operational interfaces.

Structuring the problem in TINA way had as a first step identification of TINA reference points. It turned out that it was possible to formulate the question in this way, the following figure shows the reference points in an application where third party (3pty) video-on-demand service is subscribed by a customer from a retailer. A stream interface is between the customer and the 3Pty. It realises TINA reference points TCon and ConS.

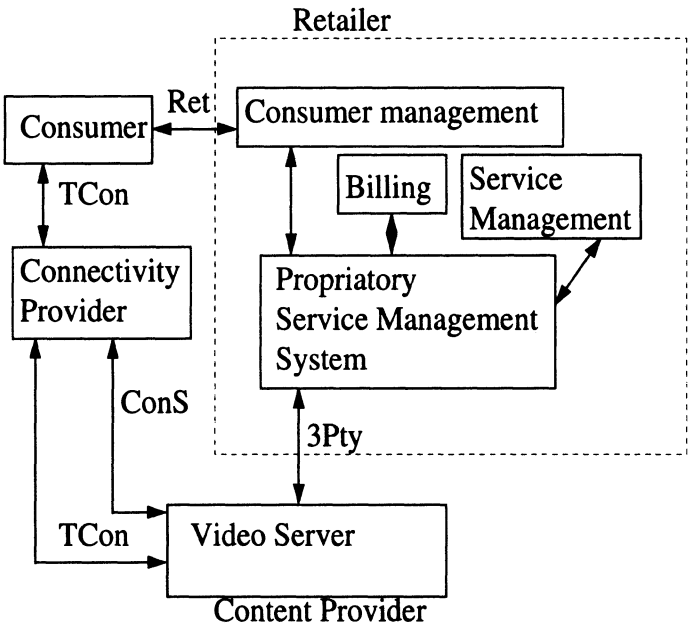

Figure 1. Initial assignment of TINA Reference points to a practical implementation.

Under CORBA, actually between two ORBs, is GIOP which now contains an operational interface for transporting signalling messages and a stream interface for transporting video and audio. These protocols must be selected. In literature there are some proposals, e.g., [9] proposes using CORBA either on SS7 or on TCP/IP where TCP/IP is on top of ATM. For mapping of CORBA to SS7 [9] gives two proposals: GIOP over TCAP and GIOP over SCCP. However, the selected solution was more practical: for signalling is used IIOP since it exists in CORBA 2.0. For streams is used a socket interface which is wrapped into objects in order to make something resembling TINA stream objects. So, both stream and operational protocols use IP.

One possibility is that the IP runs on the public Internet. This solution was tried and the delays obtained from Orbix grid example server were on the range of 1 second from Finland to Germany.

One alternative is that IP actually runs on N-ISDN in the following configuration:

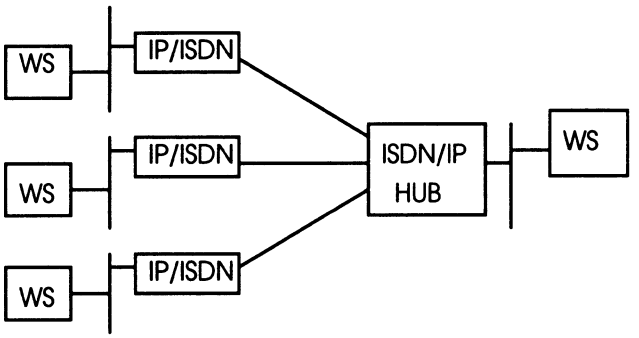

Figure 2. Stream objects managed by CORBA based IDL-defined interface and the stream is on ISDN and for the workstation ORB-software it looks like the stream is transmitted on IP (IP/ISDN router is used).

It is also possible to support several networks and have a decision point which selects the network to be used during a connection.

3. PASSING QOS REQUIREMENTS THROUGH CORBA

In this section QoS does not have the usual meaning from E.800 where QoS contains the engineering aspects of QoS, largely covered by GOS, reliability concepts and nonengineering related concepts of goodness of the service (time to repair, customer service, etc.). Instead QoS here means stream quality as in ATM when ATM cells are used to carry video and voice. This QoS concept is related to the PSTN concept of transmission quality (attenuation, jitter etc.). It is also related to the QoS parameters in many data communication protocols, e.g., OSI-session.

Inside ORB QoS can be assumed to be good but between ORBs it is determined by the stream transmission protocol. Here it is unnecessary to decide if the streams are transmitted using CORBA implying that between ORBs the streams go through GIOP or if a special protocol is transmitting the streams. It is known that the CORBA alternative has a considerable overhead and the likely choice is that the streams do not go through GIOP. Four our purposes this makes little difference since it is a question of passing the QoS parameters.

OMG has not done much work on this issue. TINA-C has treated QoS, referred to as flowQoS in [7], means the quality of streams in terms of bandwidth, jitter and errors and it is quite analogous to the QoS in ATM. The simplest solution is to use the

flowQoS concept from TINA-C but the problematic is more difficult since the realistic network alternatives for carrying streams do not use QoS in a similar way.

In ATM QoS is the quality of the connection in terms of cell losses and cell delays (CLR, CDV, CTD). It is requested by the initiator of the connection.

The concept of QoS in the Internet draft protocol RSVP is in some respect different, like the QoS is requested by the receiver and the state is soft, i.e., if the state of the sender is not refreshed regularly it is not kept.

There are also ways where QoS requirements are not passed to the stream transmission protocol but instead this lost information of needed QoS is recovered in the network/ link layer, like in different forms of IP switching.

Here we come to the same problem which the CORBA interface seemed to remove, we would need to know the underlying network in order to give a suitable QoS descriptor. Two solutions seem possible. The first is to use a QoS descriptor of undefined type (Any) and have the content the specific QoS description method of the network. This solution is easy for CORBA but makes the application network dependent. The better possibility is to classify the traffic sources by a simple integer, like {normal videophone, good video-on-demand, . . . , other} and collect statistics by a suitable statistics object which is on CORBA level. This statistics is used for mapping the traffic characteristics to QoS and traffic parameters. In this case the application only gives the classifying integer which is reserved for this purpose and not for instance a TCP-port number which has other usage.

4. IMPACT OF THE CORBA LAYER ON PERFORMANCE

As the streams are used to carry video and audio one natural reference point is PSTN where circuit-switched network is used to carry the voice and the SS7 network makes the signalling. This network has evolved to ISDN and to IN. In PSTN performance questions contain as an essential part GOS (Grade of Service) which originally meant call blocking but also contains different forms of signalling delays and blocking.

The performance impact of CORBA in a usage where the connection management of stream objects is made with CORBA should contain these GOS related matters:

identification of GOS parameters,
setting target values, the target values can be compared to those of SS7, like 100 ms packet delays,
measuring the delays, the measurement types could be described as in E.502,

breaking the delays to components, the delay components caused by CORBA being interesting here,
maybe also implementing and trying performance improvements.

Another aspect of performance impact of CORBA is the load on network elements. It this case the relevant matters are identification of:
workloads in the workstations (CPU load), throughput,
other resources in the workstations (memory usage, number of processes, number of sockets, etc.),
workloads in the network (message lengths, number of messages).

Delays in CORBA originate from several sources. The following delays also mean workloads to the workstation:
additional remote invocations for naming,
marshalling/demarshalling overhead, in OSI terminology this is encoding/decoding to transfer syntax,
demultiplexing, in OSI terminology this is address resolution,
and naturally also data copying and memory management.

Some delays in CORBA do not involve a workload. An example is invoking blocking operations which stops the invoker until there comes an answer. There are ways to avoid these delays: a nonblocking invocation can be defined in IDL by the keyword *oneway*. Also Event service enables asynchronous communication between objects.

Blocking problems can be caused in CORBA by invoking a remote object which cannot be started since there are too many process invocations or is a server supports only one thread.

5. RELIABILITY/ROBUSTNESS ISSUES

CORBA objects appear as local objects because the client's IDL stub acts as a local proxy. In reality the application is distributed, therefore reliability problems are caused by network failures. What happens in this kind of situation is that a CORBA object (a client or an server object) should receive an exception.

Robustness issues include also load balancing in case of server failure and mirrowing objects.

CORBA leaves reliability/robustness issues mostly on the responsibility of the application developer. Measurement of the reliability/robustness issues in the measurement set-up in Figure 2 is not possible in the sense that any failure probabilities

could be obtained. Instead a list of reliability/robustness issues can be composed and solutions can be implemented and tried.

6. TRAFFIC AND CONGESTION CONTROLS

There seems to be a potential problem that in CORBA architecture there can appear focused overload if a large number of clients try to use the same server object. Focused overload can appear in :

Naming, if locator of objects becomes congested since too many clients try to access objects in the same ORB. The objects need not be the same,

LifeLong, if in creating an object a loader becomes congested. Loader is involved in starting an instance of an object,

Event, if too many operations are made to an object from several clients

Solving the problem of focused overloads involves:
identification of workloads,
modelling of network elements,
defining congestion control mechanisms.

CORBA architecture does not contain traffic and congestion controls. Since there are no traffic and congestion controls, there is no reason to investigate the typical problems of such methods: fairness and possible break down of the control e.g., by spreading of overloads. Below are some suggestions.

Focused overload can arise either in the object or in the locator trying to find the object. One way to solve this is by inserting congestion control filter to the server object. Orbix CORBA software provides hooks called filters which give the implementor eight places where to insert congestion control filter code. This solution will still be sensitive to focused overloads as the server object is a centralised element. Another possibility is to insert congestion control to a local part of the server, in this case a smart proxy in Orbix terminology, However, setting parameters of the congestion control would involve asking the congestion state of the server.

Focused overloads could be considered in situations analogous to IN: overload arising from many calls passing a central element, e.g., in number translation service, overload caused by mass calls, overload caused by number portability.

7. SUGGESTED GOS/QOS PARAMETERS

QoS parameters are set for a service. In the application of CORBA to voice/video telephony there are three natural levels that could be investigated.

- On the highest level there is the TINA stream object, this object can in some way be compared to DSS1 service and the GOS parameters can be similar to call set-up time, disconnection time, call blocking.

- On the next level, this TINA stream object is made using CORBA services. Delays of the relevant CORBA services can be taken as QoS parameters.

- Below CORBA there is GIOP. The incremental delays created by GIOP can be taken as QoS parameters.

Directly using the two latter choices would involve selecting some benchmarking applications implemented on top of CORBA. There are such measurements in [1]-[6]. Using the first alternative sets already as the benchmark the implementation of the TINA stream object. The two lower levels can be treated by breaking the measured delay to actions, for instance in invocation of a remote object there are parts such as locating the server, creating an instance of the object involving memory allocation, binding to the object etc. It is also relevant to identify in each case if the delay is of latency type, like latency in a buffer, or if it presents a workload to some processing element.

7.1 GOS/QoS and NP parameters for a TINA stream computational object

In this case the interface which is considered sees the stream objects.

We cannot measure call blocking in the measurement set-up. The following GOS parameters are proposed:

ConnectionSet-upDelay
: This from a user of the client side stream object initiating a video connection to the time that the there is a one-way or two-way video established between the client and the server.

DisconnectionDelay
: This from a user of the client side stream object closing a video connection to the time that the client and the server are in the final stage of disconnection.

MessageResponseTimes
: These are response times of messages sent during the connection. The messages are not decided but could contain for example the following.

  PauseResponseTime
  : Time from sending a pause to the time the stream stops.

For Network Performance (NP) the following elements can be measured:

Message load

Separately on the stream and operational networks. In Figure 2 both networks are IP. This means that the port numbers have to be identified.

Calls per hour, transactions per hour

Maximum throughput in terms of video calls or messages.

7.2 GOS parameters for CORBA services

The CORBA services of interest for performance measurements are:

LifeLong LifeLong includes creation, moving and deletion of objects.

Naming Naming involves locating the object and binding to it.

Event Event includes RPC calls and exceptions in case of failure.

Other COSS services can be included in case they are implemented by many of used ORBs.

GOS parameters for LifeLong:

ObjectCreationTime

The time to create an object.

This operation is announced to take 650 ms in Orbix [1].

ObjectDeletionTime

The time to delete an object.

ObjectMoveTime

The time to move an object.

GOS parameters for Naming:

(Notice that _bind() is Orbix-specific, CORBA 2.0 uses factory for this purpose.)

ObjectActivationDelay

The time for remotely activating a server when the locator is not used.

In Orbix remotely activating a server when the locator service is not used has been announced in [1] to take 4 s. Activation involves that the daemon orbixd forks up a new server process and the new server makes initialisation and a call impl_is_ready.

ObjectBindDelay

The time for returning an object reference from the start of name binding.

Measurement of the Bind delay

The delay of this operation can be measured by calculating the time how long _bind() takes. _bind() can accept a server name or it can also locate the object. In [1] binding is announced to take 960 ms provided that the server is already activated.

BindException Delay

The time from sending a try of bind to receiving the exception to the CATCH in the client assuming that the server is not available.

GOS parameters for Event

EventResponseTime

The time from sending a try to getting a response (CATCH) for a selected benchmarking event.

EventExceptionTime

The time from sending a try of an event to receiving the exception to the CATCH in the client assuming that the event fails.

7.3 GOS parameters for GIOP

The measurement of delays is made between two different OBRs using IIOP where the TCP/IP actually is replaced by ISDN in the following configuration from Figure 2:

WS - LAN-IP/ISDN router - HUB - IP/ISDN router - LAN -WS

The time is measured by calling time stamps of the operating system and by a sniffer in the local LAN.

Incremental DII delay

DII presents an API that can be called from a programming language. DII can be made using Any class which can potentially lead to long delays. In this measurement the incremental delay of a benchmark implementation of a client with DII and a similar client with IDL is evaluated. What is being measured includes the delay caused by serialisation of data in IIOP (in CORBA parlance marshalling/ demarshalling). A separate measurement of this data serialisation delay is probably not feasible.

Measurement set-up for measurement of the Incremental delay of DII

Client uses DII, server object uses DSI. Client requests an attribute defined as Any. It is necessary to agree on an implementation for benchmarking.

IONA Orbix performance measurements in [1] contain some results with the following set-up:
- two lightly loaded Spark stations
- IIOP they use TCP/IP and XDR encoding.
In request/reply the delays are 8-191 ms and with one-way calls 3-140 ms.

Incremental connection set-up delay of IIOP:

Time for setting up the TCP/IP connection in the measurement set-up (Figure 2). A similar connection set-up time using directly TCP/IP through the socket interface is measured and the incremental time is attributed to CORBA.

Measurement set-up for the delay of IIOP

Client uses IDL stub, server uses IDL skeleton.

Measurement set-up for the acceptable delay of IIOP

Client uses IDL stub, server uses IDL skeleton. Introduce additional delays using a measurement set-up in order to set target QoS parameter values (maximum acceptable delays, ignorable delays).

ImplementationRepositoryResponseTime
The operation get implementation returns object implementation from the ImplementationRepository. This delay is measured by selecting a suitable benchmark implementation and measuring the response time.

InterfaceRepositoryResponseTime
The operation get interface returns interface from the InterfaceRepository. This delay is measured by selecting a suitable benchmark implementation and measuring the response time.

8. POSSIBLE METHODS FOR PERFORMANCE IMPROVEMENTS

Several performance improvement alternatives for CORBA applications have been proposed in the literature. The following list gives a few possibilities which can be or have been implemented:

- Using IIOP performance gives worse performance than using directly TCP/UDP-sockets because of CORBA overhead. CORBA can have better performance than a socket based solution if both client and server are local and ORB can dispense with interprocess communication. In [1] there is a measurement of this kind. If the server and the client share a common library is used, i.e., C-structures are passed by pointer in the same address space, the delay is 2.7 ms. It can be noted that author's solution achieving a similar performance gain in one OSI application (in VTT's X.500 a DUA communicated with a local DSA through shared memory skipping the OSI-stack) required code writing and resulted to minor differences in behaviour when using the stack and when using the shared memory. This is clearly much easier to do with CORBA.
- Another way of improving performance is caching data in a smart proxy.
- In ORBline CORBA request reuse improves performance of DII.
- Using improved elements (loader, locator, Basic Object Adaptor/ Object Oriented Object Adaptor, etc.).
- If a server object in CORBA has only one thread, only one call can be processing at a time. This can create a performance problem. Multi-thread support in servers is one solution. CORBA v2.0 does not require multi-threading. However, HP ORB plus and Multi-Threaded Orbix support this.
- Some ORB implementations contain some performance bottlenecks, e.g., Orbix using IIOP opens a new TCP/IP connection and thus a new socket for every object reference in the server side. In some other ORBs this is solved in a better way.
- There are a number of improvements to reliability, e.g., a smart proxy binds to another server if a server fails.
- ORBs can regularly bing each other and measure response times for GIOP but such method has to be built on top of CORBA.

9. RELATED WORK

In CORBA performance measurements the emphasis has been on throughput and latency, the latter term actually refers to response times. References [1]-[5] contain some results of performance measurements of CORBA. The performance studies of CORBA are similar to performance studies of distributed databases in the sense that they try to set benchmarks.

OMG has considered defining QoS for streams for CORBA [8]. These ideas are influenced by TINA-C.

10. REFERENCES:

[1] Orbix: The Orbix Architecture, IONA Technologies Ltd. August 1993. ARCH.DOC, info@iona.ie

[2] D.C.Schmidt, T.Harrison, E.Al-Shaer: Object-Oriented Components for High-speed Network Programming, USENIX, Monterey, June 1995.

[3] A.Gokhale, D.C.Schmidt: Measuring the Performance of Communication Middleware on High-Speed Networks, SIGCOMM, Stanford Univ, August, 1996.

[4] A.Gokhale, D.C.Schmidt: Evaluating CORBA Latency and Scalability Over High-Speed ATM Networks, ICDCS 97, Baltimore, May 27-30, 1997.

[5] A.Gokhale, D.C.Schmidt: The Performance of the CORBA Dynamic Invocation Interface and Dynamic Skeleton Interface over High-Speed ATM Networks, GLOBECOM, London, November 18-22, 1996.

[6] I.Pyarali, T.Harrison, D.C.Schmidt: Design and Performance of an Object-Oriented Framework for High-Speed Electronic Mefical Imaging, USENIX COOTS, Toronto, June 1996.

[7] P.Leydekkers, M.Jorgensen (ed.): Unification of Connection/Session Graphs, Stream Interfaces, and Channel Models, TINA-C Report, version 1.3, 16. April 1996.

[8] Streams and QoS: White Paper, OMG, at //www.omg.org/

[9] EURESCOM project P508, project information, available for EURESCOM partners.

[10] A.A.Lazar, S. Bhonsle, K.S.Lim: A Binding Architecture for Multimedia Networks, J. Parallel and Distributed Systems, Vol. 30, No 2, Nov. 1995, pp. 204-216, http://www.ctr.columbia.edu/comet/xbind/