PERGAMON

Computers & Graphics 27 (2003) 735–745

COMPUTERS & GRAPHICS

www.elsevier.com/locate/cag

Technical section

Interactive multimedia streams in distributed applications

Edouard Lamboray[a,*], Aaron Zollinger[a], Oliver G. Staadt[b], Markus Gross[a]

[a] *Computer Science Department, Institute of Scientific Computing, IFW D29.1, ETH-Zentrum, CH-8092 Zurich, Switzerland*
[b] *Computer Science Department, University of California, Davis, USA*

Abstract

Distributed multimedia applications typically handle two different types of communication: request/reply interaction for control information as well as real-time streaming data. The CORBA *Audio/Video Streaming Service* provides a promising framework for the efficient development of such applications. In this paper, we discuss the CORBA-based design and implementation of *Campus TV*, a distributed television studio architecture. We analyze the performance of our test application with respect to different configurations. We especially investigate interaction delays, i.e., the latencies that occur between issuing a CORBA request and receiving the first video frame corresponding to the new mode. Our analysis confirms that the interaction delay can be reasonably bounded for UDP and RTP. In order to provide results which are independent from coding schemes, we do not take into account any media specific compression issues. Hence, our results help to make essential design decisions while developing interactive multimedia applications in general, involving e.g., distributed synthetic image data, or augmented and virtual reality.
© 2003 Elsevier Ltd. All rights reserved.

Keywords: Distributed systems; Augmented, and virtual realities; Computer conferencing, teleconferencing, and videoconferencing; Performance of systems

1. Introduction

Distributed component architectures leverage the development of networked applications since they provide a powerful programming model for remote method invocation. However, distributed multimedia applications require additional features which allow for the efficient transmission of real-time streaming data.

During the last years, CORBA has been used in many research and industrial projects and has become one of the foremost standards in distributed computing [1]. Stable implementations are widely available, both as open-source and commercial software. However, CORBA was not suitable for time- and performance-critical applications, until the *Real-Time* CORBA specification was released [2].

This paper analyzes the TAO/ACE framework [3] and its CORBA *Audio/Video Streaming Service* implementation in the context of dynamically changing real-time transmissions [4]. We built an A/V streaming application as a test bed for the TAO/ACE toolkit and we benchmarked TAO's CORBA implementation using our test application. The major contribution of this work consists in the measurement of the latencies that occur between the invocation of a CORBA request and the reception of the first corresponding video frame. Our quantitative results confirm that these latencies are bounded when the streaming data is transmitted by unreliable transport protocols, such as UDP and RTP. The absolute value of the latency depends on the average network load. Furthermore, we assess the scalability of our test application with respect to the number of simultaneous video streams. Unlike existing applications, the Campus TV test bed handles a large number of high-quality videostreams with little processing overhead due to coding. Hence, it allows for the efficient testing of the raw network and data transmission aspects of the CORBA A/V Streaming Service. Thus, our

*Corresponding author. Tel.: +41-1-632-07-83; fax: +41–1-632-1596.

E-mail addresses: lamboray@inf.ethz.ch (E. Lamboray), zollinger@inf.ethz.ch (A. Zollinger), staadt@cs.ucdavis.edu (O.G. Staadt), grossm@inf.ethz.ch (M. Gross).

0097-8493/$ - see front matter © 2003 Elsevier Ltd. All rights reserved.
doi:10.1016/S0097-8493(03)00147-X

analysis is valid not only for video, but for all type of streamed real-time data, including computer generated image data.

Our study of CORBA's suitability for distributed and interactive multimedia streaming is part of the *blue-c* project, whose aim is the development of a novel platform for highly immersive collaborative virtual environments [5] (http://blue-c.ethz.ch). It comprises, among other things, real-time acquisition of several video images of a human user and reconstruction of a 3-D graphical representation. This *geometry-enhanced video* will be transmitted via a high-speed network to several blue-c portals, where it will be seamlessly integrated into a distributed virtual environment. The blue-c system will eventually lead to a distributed virtual reality platform on which collaboration is possible using the most natural ways of interhuman communication and interaction, enhanced through the feeling of total immersion. Several research groups at ETH Zurich currently develop the various blue-c hard- and software components.

The blue-c environment will be a heterogeneous system on different platforms, including SGI Irix, Linux and Microsoft Windows. The increasing complexity of this system demands a middleware with an advanced programming model for distributed computing, supporting portability and real-time features. The chosen middleware needs to provide an efficient way of transmitting both control information, allowing for dynamic configuration of the system's setup and the exchange of less time-critical information, as well as latency-critical real-time streams with bandwidth requirements up to several megabits per second. From this point of view, the blue-c platform is not much different from other distributed applications which share high-quality multimedia streams.

From a practical point-of-view, Campus TV, our test bed, implements a distributed television studio which is depicted in Fig. 1. The following components are connected in a local area network:

- A variable number of *camera stations* that acquire audio and video signals and continuously transmit them to a control desk.
- A *control desk*, providing a graphical user interface which allows an operator to preview and configure all available streams.
- A theoretically unlimited number of *receivers* that listen to the multicast address of the TV program.

The remainder of this paper is organized as follows. After reviewing CORBA and its use in multimedia applications in Section 2 and related work in Section 3, we describe the overall architecture of our application in Section 4. Section 5 discusses some implementation details of the different components and Section 6 presents our performance analysis.

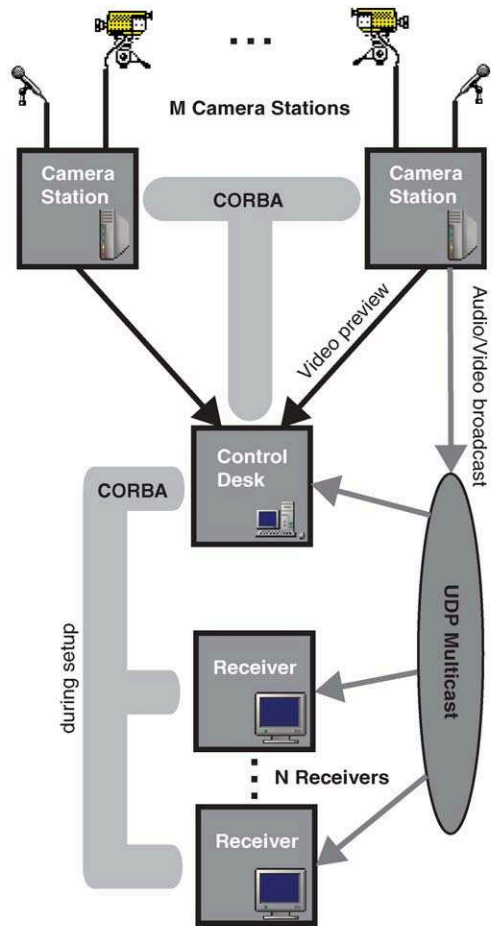


Fig. 1. Campus TV, a distributed TV studio architecture.

2. CORBA and multimedia

A lot of effort has recently been put into the development of general-purpose communication middleware. A middleware is an intermediate software layer between low-level application programming interfaces and application code. The use of middleware toolkits leads generally to a better structured software architecture and allows for easier portability from one platform to another. However, not every middleware standard is appropriate for building distributed multimedia applications. The special requirements of this type of applications can be summarized as follows:

- Handling of continuous media streams.
- Support for multi-party communication.
- Quality of Service management.
- Real-time synchronization.

Classical environments for distributed computing often do not fulfill the requirements of distributed multimedia applications [6]. Their initial focus lies in remote request/reply object interactions and their capabilities in continuous media modeling are limited [7]. Also the *Common Object Request Broker Architecture* (CORBA), supported by the Object Management Group (OMG) [1], does not naturally support the transmission of real-time streaming data. The CORBA layer introduces a large overhead through its marshalling operations and through the retransmission of lost packets [3,8].

CORBA's *Audio/Video Streaming Service*, specified in [4], uses a promising concept for the transmission of time-critical streaming data. It makes a distinction between control information (i.e., connection setup, device configuration) and the real-time payload data. The control information can perfectly be handled by CORBA's client-server programming model, but the real-time data is directly streamed using classical transport protocols. On one hand, this design profits from advantages of the CORBA programming environment, on the other hand, it produces no overhead for the time-critical data. In order to guarantee interoperability between different A/V streaming applications, the service defines common interfaces for control and management of streams. Moreover, each stream can have an associated media controller, which implements application specific functionality. The A/V Streaming Service specification does not prescribe interface details of media controllers, such that they can be flexibly designed by the application developers.

A schematic description of the A/V Service is depicted in Fig. 2. A stream contains one or many data flows and connects as many data sources with their corresponding data sinks. An endpoint, represented by the `StreamEndPoint` interface, comprises:

- A flow endpoint, which is either a data source or a data sink, represented by the `FlowEndPoint` interface.
- A stream interface control object, represented by the `StreamCtrl` interface, providing an IDL defined interface for controlling and managing the stream.
- A stream adapter, that receives or transmits data frames over a network.

3. Related work

The Center for Distributed Object Computing at Washington University, St. Louis, provides with the TAO/ACE framework an advanced CORBA implementation that includes real-time features and additional services of CORBA 2.x, including the A/V Streaming Service [8]. ACE, the *Adaptive Communication Environment*, is an open-source object-oriented C++ framework that implements many core design patterns for concurrent communication software [9]. Furthermore, it provides an operating system abstraction layer and therefore improves the portability of applications that are built upon ACE. *The ACE ORB* (TAO) implements the standard CORBA reference model with most of the enhancements for real-time applications. Further information about TAO/ACE, as well as the sources of the current version can be downloaded at http://www.cs.wustl.edu/~schmidt/TAO.html. An MPEG video server is included as an A/V Streaming Service example application in the TAO/ACE distribution. It is

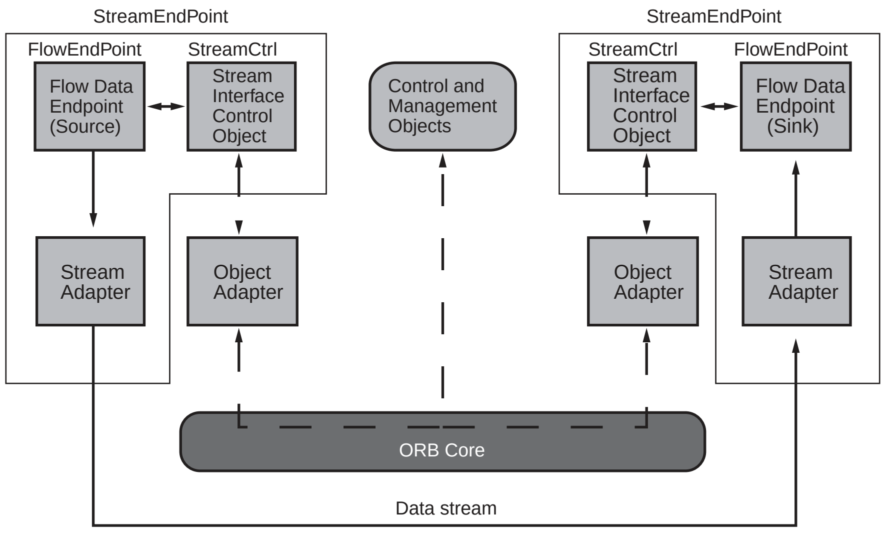


Fig. 2. Data transmission using CORBA's audio/video streaming service.

based on the *Distributed A/V MPEG Player* from the Oregon Graduate Institute [10]. The MBone video-conferencing application *vic* [11] was also ported to TAO's A/V Streaming Service [8]. Recent work includes a video distribution application, combining the A/V Streaming Service with the Quality Objects framework [12]. None of these applications allows a simple adaptation for a performance analysis similar to the tests we present in Section 6.

In the field of collaborative virtual reality and distributed simulation, design suggestions for CORBA-based frameworks have been made [13–15] and on-going projects exist [16,17], but no complete frame-work has been implemented until today.

ISO published through the *Reference Model for Open Distributed Processing* (RM-ODP) a meta-standard for distributed processing [6,18]. CORBA, as well as other concrete standards, like PREMO (PResentation Environments for Multimedia Objects) [19], implement the concepts of RM-ODP up to various extents. Further attempts to integrate CORBA with multimedia processing and transmission can be found in [20,21]. At the time of those research projects, the CORBA A/V Streaming Service was not yet fully specified.

The *Real-time Transport Protocol* (RTP) is an appropriate transport protocol for multimedia data delivery [22], it will be taken into account in Section 6 of this paper.

McCanne et al., propose a common infrastructure for multimedia networking middleware, the *MASH* project [23]. Their research effort generated a large number of applications, among which a video conferencing application that can be remotely controlled [24], and a control system for live webcasts [25]. The webcast application, which has a more developed functionality than Campus TV, currently still requires special hardware for video switching and is implemented using ad hoc remote method invocation.

4. System description

This section gives an overview of the Campus TV test application we built for the evaluation of the CORBA A/V Streaming Service. The supported data classes are described, together with some important design decisions.

4.1. Overview

In our application, several camera stations grab live video images, potentially at a low resolution and at a low frame-rate. This information is continuously streamed to the control desk. A user can select the camera that should send its image as the active image to the receivers. In the following, we will refer to the active camera station as the broadcast camera station. We use this term from the area of television transmission, but it should not be mixed up with its meaning in the context of computer networks.

The choice of the broadcast station is communicated to the camera station using CORBA requests, whereas the video and audio data are distributed using configurable IP transport protocols. The camera station which gets the broadcast token from the control desk starts sending its input in high resolution and at highest achievable frame-rate to a multicast address. All control information, like image dimensions and frame-rate, can be dynamically configured during a session. Changes in the setup are also propagated using the ORB. Fig. 1 shows a system overview of our test bed.

The receivers first ask the control desk for information about the currently distributed program. Then they add themselves to the multicast session using the A/V Streaming Service. The control desk knows about all the distributed camera stations delivering possible input. But it does not keep track of the receivers and does not exchange any information with them, except for the setup data.

Since the high-quality stream of the broadcast camera station is distributed using UDP multicasting, the system scales very well to a high number of receivers. Furthermore, there is no a priori limit on M, the number of camera stations, even though every new camera station introduces additional network traffic and increases the workload at the control desk. In fact, we can trade-off M against the quality of the preview images, while dynamically configuring the image resolution and the frame-rate of the preview video streams. Finally, since there is no centralized sender of the broadcast stream, the A/V data is transmitted with minimal overhead and latency.

4.2. Supported data

4.2.1. Control information

The control information, which is different from the audio and video samples, includes commands for managing the streams, i.e., start/stop, as well as configuration parameters for the data acquisition. It can also be used for monitoring the system. The control desk configures for example:

- The dimensions of the video images and the acquisition frame-rate at the camera station.
- The sample frequency of the audio acquisition.
- The permission to send to the multicast address.

The action of switching from camera station A to camera B station as the broadcast station can thus be summarized with the following actions:

- Send current broadcast settings to *B*.
- Instruct *A* to stop sending to the multicast address.
- Instruct *B* to start sending to the multicast address.

The A/V Streaming specification suggests that a media control object is associated to each `FlowEndPoint`. The media controller implements device specific operations, which are not general enough to be handled by `StreamCtrl` objects. Since the media controller is completely specified by the application developer, any type of information can be communicated and the requirements of different applications can be met. Fig. 3 depicts a typical stream setup in a point-to-point connection and shows the different paths for control and real-time streaming data.

4.2.2. *Audio/video transmission*

The video frames are grabbed as RGB images with 24 bits per pixel color information. As camera stations, we use SGI O2 workstations with video option. In this paper, we focus on dynamically configurable A/V streams and we assess the performance of TAO's A/V Streaming Service implementation. Hence, we did not yet integrate any compression algorithms in Campus TV. Of course, this would be necessary for deploying Campus TV in a large scale environment.

TAO's A/V Streaming Service implementation provides a pluggable protocol framework, in which the most common IP-based transport protocols, such as TCP, UDP (unicast and multicast), and RTP, as well as ATM, are already included. A factory object encapsulates the concrete protocol objects and simplifies the addition of future data transfer protocols. Furthermore, it allows us to rapidly integrate different protocols into our application. Since the connection setup is completely handled by the A/V Streaming Service specification, the application programmer only needs to adapt his payload data to the various protocols. In our case, we use for example sequence numbers for identifying the fragmented image packets. TCP and RTP have already sequence numbers in their protocol headers, UDP has not. Hence, we implemented an additional software layer, that handles the protocol-dependent packetization of our pay-load data.

The CORBA A/V specification distinguishes between flows, carrying data in one direction, and streams, comprising a set of flows. There are two ways of specifying a connection, either via the *light* or via the *full profile*. In the light profile, the application is granted access only to stream endpoints, as well as to virtual and multimedia devices. According to [4], virtual and multimedia devices abstract physical or logical devices consuming or producing the samples of the multimedia stream. If access to the flow end-points or the flow connection is required, the full profile must be used. The TAO/ACE framework implements both configurations. We used the full profile, since the separate access to the video and audio flows makes our design more flexible for further investigations.

A critical issue for the efficient distribution of data from one to many users is the availability of point-to-multipoint communication. Of course, a point-to-multipoint distribution can be realized using many point-to-point connections, but this strategy does not make efficient use of the available resources. The CORBA A/V Service specification includes point-to-multipoint binding, but does not define multipoint-to-multipoint binding, where several sources communicate with several sinks. The TAO framework, however, supports multipoint-to-multipoint communication, which finally is an important feature of our test application.

Additionally, in classical UDP multicasting, the data source is not aware of whoever is listening to its data transmission. In the CORBA A/V Service, however, each data sink needs to be explicitly added to the data sources' stream controllers. In our application, the control desk defines naturally the stream configuration and candidate camera stations as well as receivers must

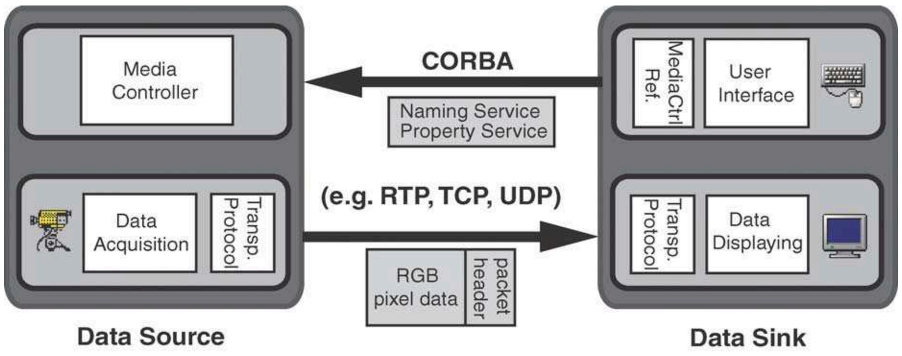


Fig. 3. Streaming video images from a data source to a data sink using a point-to-point connection.

judge during connection establishment if they are able to handle the available streams.

5. Implementation

In the following, some implementation details of the different components of our Campus TV application are presented.

5.1. Control desk

The control desk is used for configuring a broadcast session and is the interactive part in our application. A screenshot representing the control desk's graphical user interface is shown in Fig. 4. The left part of the user interface is dedicated to the thumbnail preview images, the right part deals with the broadcast stream.

The control desk first registers itself at the CORBA *Naming Service*, where the clients can later retrieve a reference to the control desk object.

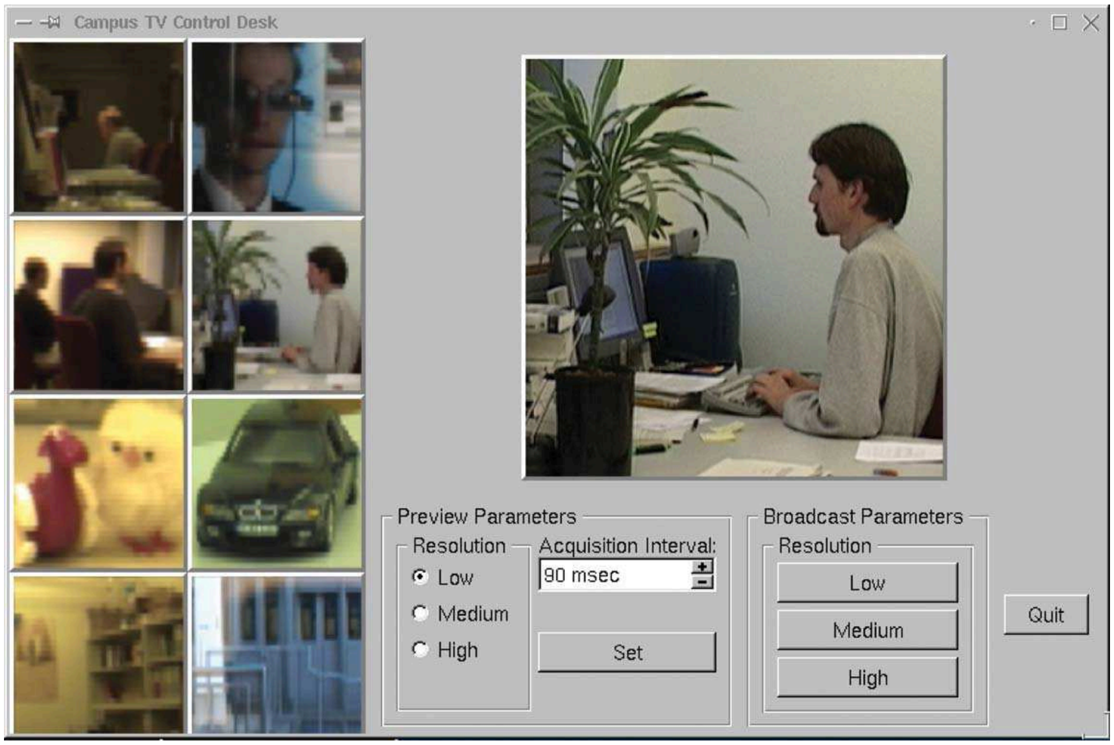
Upon the clients' requests, the control desk distributes setup information to both camera stations and receivers. From a practical point of view, a separate service could offer this functionality to the receivers.

Furthermore, the control desk initializes the multicast transmission, since it is the first listener to the attributed multicast address. It also creates the first data source, which is actually a dummy source, but which is required by TAO's implementation of the A/V Streaming specification for multipoint connection setup.

The application can be implemented according to two different strategies: *reactor-based or process-based*. For the Campus TV control desk, the process-based strategy would lead to a new process for every connecting camera station. Currently, we implemented the reactor-based strategy, i.e. all input from the camera stations is handled by the same process, and events indicate that new data has arrived. Since the update rates of the thumbnail preview images are not very high, the reactor-based control desk's performance is sufficient. An analysis of the control desk's scalability with respect to the number of camera stations can be found in Section 6.3.

Apart from the *Naming Service*, the control desk also uses the CORBA *Property Service* for retrieving a reference of the media controller of a given camera station.

All graphical user interfaces have been implemented using Trolltech's Qt cross-platform GUI framework (http://www.trolltech.com). Since our goal is to build a portable communication layer, we also need a GUI toolkit available for all platforms of interest. Another significant advantage of the Qt library consists in its



Resolution: [✓ Low] Medium High

Acquisition Interval: 90 msec

Fig. 4. The graphical user interface of the control desk.

OpenGL support (http://www.opengl.org). The Qt classes `QGLWidget`, `QGLContext` and `QGLFormat` allow, respectively, to create an OpenGL rendering context and display format and to render an OpenGL scene integrated into a Qt GUI. Although OpenGL rendering is not necessary for Campus TV, we will need it for our future research, which deals with geometry-enhanced video and its integration into virtual scenes.

Rendering video images in OpenGL is possible with two different strategies: The OpenGL `glDrawPixels` function allows us to write a block of pixels directly to the frame buffer. The second possibility consists in a texture map of the video image on a polygon. On modern graphics processing units, texture mapping is generally faster and less CPU-consuming than the pixel drawing strategy.

Note that the Simple DirectMedia Layer (http://www.libdsl.org) provides a similar functionality and could be used for an alternative implementation.

5.2. Camera station

The video images are grabbed at regular time intervals. The frame-rate of the video acquisition can be controlled by changing the time-out value. The audio acquisition is implemented in an analogous way to the video acquisition.

When a camera station is started, it asks for a reference to the control desk at the *Naming Service* and then connects itself with the control desk. It starts transmitting video data according to the current configuration. The camera station joins the multipoint-to-multipoint connection already during setup. Hence, it is ready to start sending both audio and video data to the multicast address as soon as it becomes the broadcasting station.

5.3. Receiver

The receiver first retrieves information about the current session from the control desk. Then it connects to the multicast address and presents the received data to the user.

Receiver and control desk only interact during the initial connection. The control desk cannot directly configure the receivers and is unaware of the number or type of current receivers. Currently, our implementation does not include an explicit synchronization for audio and video play out.

6. Performance analysis

This section summarizes the performance analysis we conducted with the Campus TV test bed. After describing the experimental setup, we discuss our results concerning interaction delays, scalability and audio/video jitter.

6.1. Experimental setup

For the following experiments, we used up to ten SGI O2 workstations with video option as possible camera stations and an SGI Octane as the control desk's host. The workstations were all running SGI Irix 6.5 and connected by a 100 Mbps Fast Ethernet switch. We ran the tests when the average load on our local area network was low.

Note that all experiments in Sections 6.2 and 6.3 were done without audio transmission. Moreover, some of the following configurations are only useful for testing purposes. Running Campus TV in "application mode" requires low resolution preview images at 5–10 Hz and high-resolution broadcast images, using RTP for data transmission.

6.2. Interaction delay

We call *interaction delay* the duration between issuing a CORBA request at the control desk and the moment of reception of the first video frame according to the new mode. In practice, our camera station's media controller implements a method for changing the color of a given pixel after frame acquisition. At the control desk, we check the test pixel's RGB values in the received video frames, and hence we can measure the delay between issuing the `setColor` command at the control desk and the reception of the corresponding frame. The IDL interface of the function is as follows:

```
struct Color {
          short r;
          short g;
          short b;
};
void setColor (in Color col);
```

We measured the interaction delay by generating automatically a series of `setColor` commands at the control desk. We fixed a range for the duration in between two `setColor` invocations and generated uniformly distributed time-outs in that range. The test results presented in this paper were obtained with time-out values between 250 and 750 ms. They are based on 1024 samples, respectively, i.e., every experiment ran for about 8 min and 30 s. We repeated the same experiments for various time-out ranges, but we did not observe significant differences in the resulting interaction delays.

Three image resolutions for both preview and broadcast images can be configured:

- low: 64×64 pixels.
- medium: 128×128 pixels.
- high: 256×256 pixels.

Table 1 shows the mean value and the standard deviation for different preview image resolutions and for UDP, RTP and TCP point-to-point connections, respectively. The resolution of the broadcast image was 256×256 pixels in each case. The interaction delay was measured on the broadcast connection. With the resolution of the preview images, we can influence the network traffic. In the low- and medium-resolution configuration, we used 10 camera stations, sending preview images at 10 Hz. This leads to a network traffic of approximately 34 or 62 Mbps, respectively. The broadcast video is always streamed at the maximum possible frame-rate. For the low and medium sized preview images, a broadcast frame-rate of 14–17 Hz is achieved. In the high-resolution case, we only used five camera stations at 10 Hz, which leads to an approximate network traffic of 100 Mbps, i.e., a certain amount of packets is certainly lost. The broadcast frame-rate drops down to 5–7 Hz. The loss indicated in Table 1 refers to the broadcast stream.

Note that in the UDP test case, we also used UDP multicasting for the multipoint-to-multipoint broadcast communication, in the RTP and TCP test cases, we used RTP multicasting for the broadcast image. A sample is regarded as lost if a corresponding broadcast image frame is never received.

In Figs. 5 and 6, the vertical axis indicates, for a time interval Δt, the percentage of samples, where the interaction delay was larger than Δt. Hence, Figs. 5 and 6 represent an approximation of a probability distribution $P(t)$, where $P(t = \Delta t)$ is the probability that the interaction delay is larger than Δt.

In Fig. 5, we observe that in the configuration with low sized preview images, 99% of the interaction delays were shorter than 175 ms. The choice of the transport protocol did not significantly influence the probability distributions. In the case of medium sized preview images, the probability distributions for UDP and RTP are only slightly different from the previous case. But for

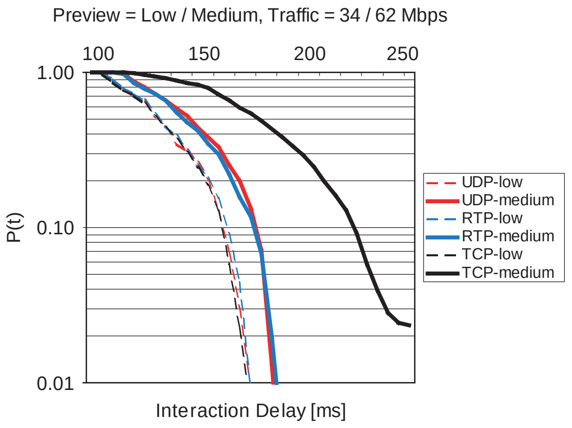


Fig. 5. Semi-logarithmic plot of the interaction delay distribution for configurations with low- and medium-resolution preview images from ten camera stations, the broadcast image has high resolution.

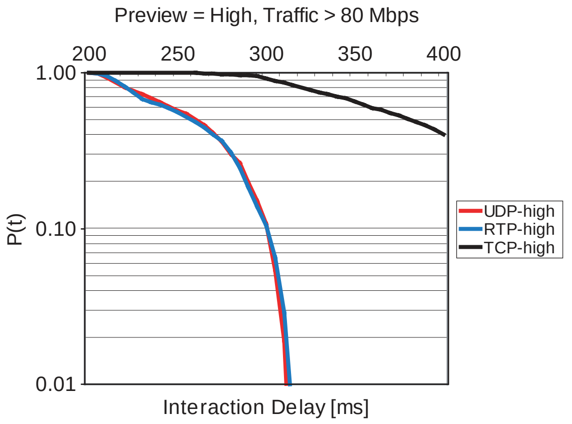


Fig. 6. Semi-logarithmic plot of the interaction delay distribution for configurations with high-resolution preview images from five camera stations, the broadcast image has high resolution.

Table 1
Interaction delay measurement statistics for different protocols and preview image resolutions, using a broadcast resolution of 256×256 pixels

Preview/broadcast	Mean (ms)	StDev (ms)	Min (ms)	Max (ms)	Loss (%)
UDP-low/UDP-high	135	20	104	179	0
RTP-low/RTP-high	136	20	104	182	0
TCP-low/RTP-high	134	19	104	183	0
UDP-medium/RTP-high	148	21	113	199	0
RTP-medium/RTP-high	146	21	101	212	0
TCP-medium/RTP-high	184	45	111	577	0
UDP-high/UDP-high	258	32	203	316	32
RTP-high/RTP-high	257	32	204	320	36
TCP-high/RTP-high	425	146	250	1074	3

TCP, we already observe a significant increase of the interaction delay.

In the case of preview images at a high resolution, see Fig. 6, the average interaction delay increases by more than 100 ms for UDP and RTP. Also the standard deviation of the UDP and RTP data sets increases by approximately 50%. We realize that in this configuration, TCP is not competitive anymore, the interaction delay values of the TCP samples are much higher than for the two unreliable protocols. Note that in this test case, where the network traffic is very high, a certain amount of multicast frames from the broadcasting camera station get lost, and hence not every sample generates a valid interaction delay. The amount of lost frames is indicated in Table 1.

Furthermore, part of the observed interaction delay is due to the low frame-rate at the high-resolution configuration. If both resolutions for preview and broadcast images are 256×256 pixels, the frame-rate of the broadcast stream drops down to 5–7 Hz. The expected delay simply due to this low frame-rate is at least 70 ms, and hence cannot be totally neglected.

The probability distribution curves of the UDP and RTP experiments lead to the conclusion that the interaction delay can be bounded. For each setting, the interaction delays occurred in a given time frame, whose mean and width were depending on the average network load. The interaction delay is much less predictable for TCP because of its inherent flow control algorithms.

Finally, we observe that the probability distributions for RTP and UDP are in general very similar. On one hand, this can be expected since RTP uses UDP as its transport protocol. On the other hand, it confirms that TAO's RTP implementation does not introduce a significant amount of overhead.

6.3. Scalability

Fig. 7 shows the average CPU load at a camera station for various configurations. The camera station which is also the broadcast station needs to transmit both low-quality preview images and high-quality broadcast images. We observe that the CPU load increases significantly with the size of the video frames. The acquisition frequency is only relevant for the preview images, the broadcast images are always streamed at the maximum possible frame-rate. Note that in the configuration with high-resolution broadcast images and low-resolution preview images, acquisition frequencies higher than 20 Hz are not achievable. The two first data sets in Fig. 7 were obtained with a preview stream only.

In Fig. 8, it appears that the CPU load at the control desk is much more influenced by the number of camera stations than by their frame-rate. If we increase the number of camera stations from one to eight, the CPU

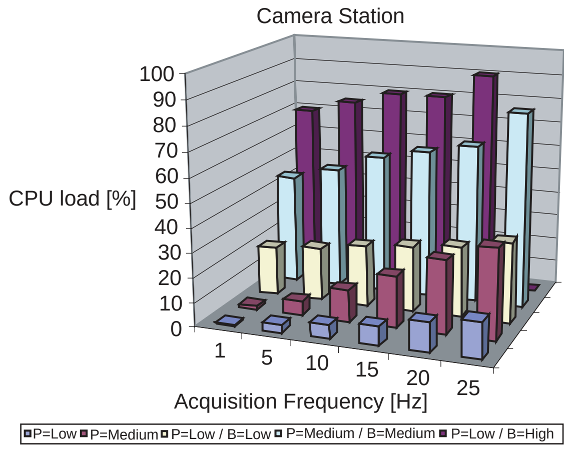


Fig. 7. CPU load for different RTP *Preview/Broadcast* configurations at the camera station.

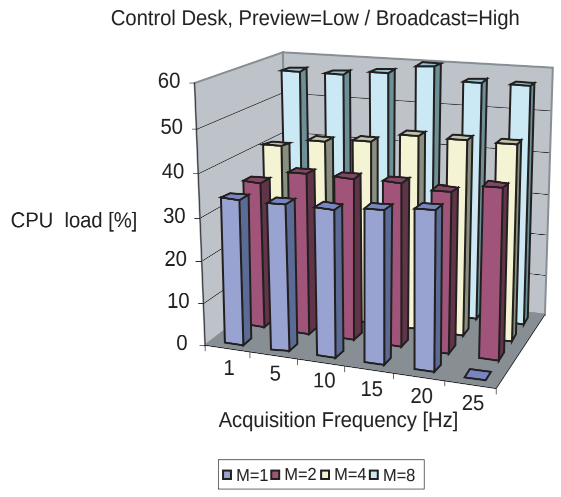


Fig. 8. CPU load at the control desk with respect to the number of preview images and preview acquisition frequency with low preview size and high-resolution broadcast using RTP.

load only grows from 30% to 60%. For high acquisition frequencies, the network load does not allow a very regular frame-rate for the broadcast stream. This explains why the variation in some test cases is not monotonic. In fact, the update rate of the broadcast image at the control desk lies between 14 and 17 Hz.

The scalability experiments were all done with RTP and RTP multicast for the preview and the broadcast images respectively.

6.4. Audio/video jitter

Additionally, we investigated the phase jitter between audio and video data. Looking at the video and audio packet transmission, we observed no delays between audio and video packets at the broadcast camera station and at the receivers. However, as can be seen in Fig. 7, the high-quality video transmission uses already 80% of the camera station's CPU. In this case, a constant delay of about 0.5 s between audio and video play out can be perceived. It can be explained by our non-optimized audio acquisition scheduling and play out algorithms. An optimization of the audio transmission was beyond the focus of this paper, since, in the blue-c system, speech transmission will be handled by a different subsystem than the geometry-enhanced video transmission.

7. Conclusions and outlook

The performance analysis presented in this paper shows that the CORBA A/V Streaming Service introduces no critical overhead for multimedia data transmission. The concept of media controllers allows the flexible integration of application specific interactions within the streaming framework. Our measurements show that the interaction delay is bounded when using appropriate transport protocols, like UDP and RTP. The absolute value of the interaction delay depends on the average network load. Furthermore, the Campus TV control desk scaled well with an increasing number of camera stations. In the future, the pluggable protocol framework of the TAO service implementation will allow us to run similar tests on different network configurations. Finally, the results of this study will influence important design decisions in our future distributed virtual reality platform, i.e. we will use our findings for deciding what type of requests can be handled by the ORB, and what type of information, apart from the multimedia data, needs to be streamed in real-time.

However, concerning the TAO/ACE framework, we encountered some problems during the implementation of Campus TV. The connection setup is not completely transparent and we had to take special care with RTP transmission. This is especially due to the fact that TAO's RTP implementation is still a bit unwieldy. But improved TAO versions appear on a regular basis and its open source character allows for bypassing local problems.

Note that in our target application, the blue-c system, the data for controlling and monitoring will be much more complex and diverse than in the Campus TV test application. At that point, we will take fully advantage of the standardized stream interfaces of the A/V Streaming Service as well as of the flexibility provided through the concept of media controllers. Possible optimizations with respect to acquisition and compression will also be taken into account for the geometry-enhanced video streams in the upcoming blue-c prototype. A first prototype is currently under development and uses extensively the functionality provided by CORBA and its A/V Streaming Service. In the future, we also want to further investigate the possibilities offered by the MPEG-4 standard, which supports the integration of multimedia streams and traditional A/V formats with synthetic images as well [26].

Finally, we envision to extend the Campus TV application such that special effects, e.g., image warps and fade-outs, can be performed on the video images. These special effects can be easily and efficiently implemented at the receiver, using OpenGL commands on the polygons on which the live video texture is mapped.

Acknowledgements

We would like to thank the numerous researchers who developed the TAO/ACE framework as an opensource project. Many thanks to the Perceptual Computing and Computer Vision Group at ETH Zurich for letting us use their hardware equipment. The blue-c project is funded by ETH Zurich as a "Polyprojekt".

References

[1] The Common Object Request Broker: Architecture and Specification. Object Management Group, Revision 2.5, September 2001.

[2] Schmidt DC, Kuhns F. An overview of the real-time CORBA specification. IEEE Comput (special issue on Object-Oriented Real-time Distributed Computing) 2000; 33(6):56–63.

[3] Schmidt DC, Levine DL, Cleeland C. Architectures and patterns for high-performance, real-time CORBA object request brokers, vol. 48 of advances in computers. New York: Academic Press; 1999. p. 1–118.

[4] Audio/Video Stream Specification. Object Management Group, January 2000.

[5] Staadt OG, Gross MH, Kunz A, Meier M. The Blue-C: Integrating real humans into a networked immersive environment. In: Proceedings of ACM Collaborative Virtual Environments 2000, San Francisco, September. New York: ACM Press; 2000. p. 201–2.

[6] Blair G, Stefani J-B. Open distributed processing and multimedia. Reading, MA: Addison-Wesley; New York: Longman; 1997.

[7] Waddington DG, Coulson G. A distributed multimedia component architecture. In: Proceedings of the First International Workshop on Enterprise Distributed Object Computing, Gold Coast, Australia, October 1997. p. 334–47.

[8] Mungee S, Surendran N, Krishnamurthy Y, Schmidt DC. The design and performance of a CORBA audio/video streaming service, Chapter in Design and Management of Multimedia Information Systems: Opportunities and Challenges. Hershey: Idea Group Publishing; 2000.

[9] Schmidt D, Stal M, Rohnert H, Buschmann F. Pattern-oriented software architecture: patterns for concurrent and networked objects, vol. 2. Wiley: New York; 2000.

[10] Cen S, Pu C, Staehli R, Walpole J. A distributed real-time MPEG video audio player. In: Proceedings of the Fifth International Workshop on Network and Operating System Support of Digital Audio and Video (NOSSDAV '95), Durham, New Hampshire, April 1995. p. 151–62.

[11] McCanne S, Jacobson V. vic: A flexible framework for packet video. In: Proceedings of the ACM Multimedia'95, San Francisco, California, November 1995. p. 511–22.

[12] Karr DA, Rodrigues C, Krishnamurthy Y, Pyarali I, Schmidt DC. Application of the QuO quality-of-service framework to a distributed video application. In: Proceedings of the Third International Symposium on Distributed Objects and Applications, Rome, Italy, September 2001.

[13] Au TA. Performance issues of HLA run time infrastructure based on CORBA. In: Proceedings of The Simulation Technology and Training (SimTecT2000) Conference, Sydney, Australia, 2000.

[14] Deriggi Jr. FV, Kubo MM, Sementille AC, Brega JRF, dos Santos SG, Kirner C. CORBA Platform as Support for Distributed Virtual Environments. In: Proceedings of the IEEE Virtual Reality 1999 Conference, March 1999. p. 8–13.

[15] O'Ryan C, Levine DL, Schmidt DC, Noseworthy JR. Applying a scalable CORBA events service to large-scale distributed interactive simulations. In: Proceedings of the Fifth Workshop on Object-oriented Real-time Dependable Systems, Montery, CA, November 1999.

[16] Gallop J, Cooper C, Johnson I, Duce D, Blair G, Coulson G, Fitzpatrick T. Structuring for extensibility—adapting the past to fit the future. In: Proceedings of CBG2000, the CSCW2000 Workshop on Component-Based Groupware, Philadelphia, Pennsylvania, USA, December 2000.

[17] Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)—Framework and Rules. IEEE Standard 1516, September 2000.

[18] Open Distributed Processing—Reference Model. Standard ISO/IEC 10746, 1995.

[19] Duke DJ, Herman I. A standard for multimedia middleware. In: Proceedings of the Sixth ACM International Conference on Multimedia, Bristol, United Kingdom, September 1998. p. 381–90.

[20] Blum C, Molva R. A CORBA-based platform for distributed multimedia applications. In: Proceedings of Multimedia Computing and Networking MMCN'97, San Jose, CA, February 1997.

[21] Yun T-H, Kong J-Y, Hong JW. A CORBA-based distributed multimedia system. In: Proceedings of the Fourth Workshop on Distributed Multimedia Systems. Vancouver, Canada, July 1997. p. 1–8.

[22] Schulzrinne H, Casner S, Frederick R, Jacobson V. RTP: A transport protocol for real-time applications. RFC 1889, January 1996.

[23] McCanne S, Brewer E, Katz R, Rowe L, Amir E, Chawathe Y, Coopersmith A, Mayer-Patel K, Raman S, Schuett A, Simpson D, Swan A, Tung T-L, Wu D, Smith B. Toward a common infrastructure for multimedia-networking middleware. In: Proceedings of Seventh International Workshop on Network and Operating Systems for Digital Audio and Video (NOSSDAV'97), May 1997. p. 41–51.

[24] Hodes T, Newman M, McCanne S, Landay J, Katz R. Shared remote control of a video conferencing application. SPIE Multimedia Comput. Networking 1999. Proceedings of SPIE, vol. 3654, p. 17–28.

[25] Yu T-P, Wu D, Meyer-Patel K, Rowe L. dc: A live webcast control system. In: Proceedings IS & T/SPIE Symposium on Electronic Imaging: Science & Technology, Multimedia Computing and Networking, San Jose, CA, January 2001.

[26] Overview of the MPEG-4 Standard. ISO/IEC JTC1/SC29/WG11 N3444, May/June 2000.