Management of Proxy Objects Providing Multimedia Applications in the Mobile Environment

J. Seitz, K. Cheverst, N. Davies, M. Ebner, A. Friday
Computing Department, Lancaster University
Lancaster LA1 4YR, U. K.
Tel. ++44 (0)1524 65201, Fax. ++44 (0)1524 593608
E-mail (seitz, kc, nigel, ebner, adrian)@comp.lancs.ac.uk

Abstract

Due to increasing computational power, long-life batteries and wireless links with improving bandwidth, distributed multimedia applications have become more and more attractive for mobile users. However, the varying quality of service (QoS) on the wireless link is still a major problem. One, generally accepted solution to this problem utilises 'proxies' which act on behalf of the mobile user and modify the communication data stream in order to match the current QoS conditions. These proxies are generally dependent on protocol layers, applications or application data. Therefore, in order to enable proxy objects to be flexibly inserted for a given multimedia stream, an architecture is required for managing supplied and installed proxy objects.

Keywords

Management of Distributed Object-Oriented Multimedia Applications, Mobile Systems, CORBA, Quality of Service, Proxy

1. Introduction

Multimedia communication has been made feasible by emerging broadband networking technologies based on fiber, e.g. ATM, and high performance chips. Furthermore, due to increasing battery power and improving display technology, mobile computers have become more and more popular. However, new wireless "untethered" communication services must be able to adapt to a constantly changing communications environment brought on by mobility [12]. Because physical problems like signal reflection, fading and distortion prevent a constant and high bit rate over a wireless link, the strong QoS requirements of multimedia applications have to be decreased by modifying the communication stream to adapt its requirements to match the characteristics of the wireless transmission link.

One way to achieve this adaptation is to install *proxies* into the communication data stream. A proxy acts on behalf of the mobile client, separating the wireless from the wireline link. In addition to this, it also acts as a mediator between these two different links modifying the communication stream to adapt its requirements

(see figure 1, [20]). The technique to insert a proxy has been most commonly used to access World-Wide Web pages [15] and to implement firewalls [14]. However, the idea of proxies has proved to be valuable for mobile clients, too.

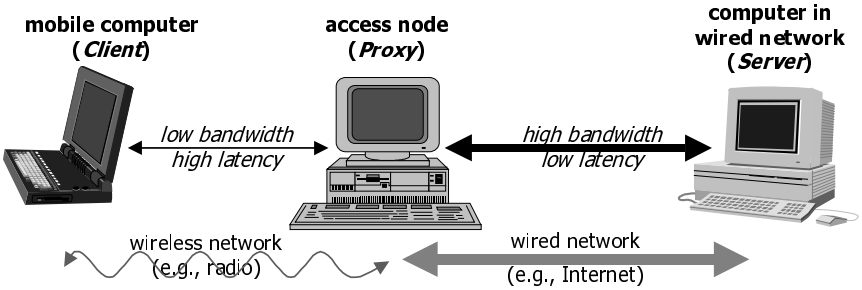

Figure 1: Scheme of a Proxy

However, although this approach has been implemented in several research projects, the insertion of proxies remains rather static. Thus, we propose an architecture to flexibly insert proxies "on the fly". The architecture is based on the *Common Object Request Broker Architecture (CORBA)* standard by the Object Management Group [16]. CORBA is a widespread architecture for the implementation of distributed applications. Its applicability in the area of (tele-)communication services and their management has also been proved [21]. Furthermore, actual research projects deal with real-time CORBA implementations.

This paper focuses on the management of proxies. For selecting and installing proxies, a generic modeling technique has to be defined. We propose a multi-layered technique to supply several interfaces that allow the use of the proxies while assuming different knowledge of their functionality. Furthermore, we detail the part of the architecture dealing with supplying, selecting and installing proxies.

The remainder of this paper is structured as follows. Section 2 introduces the RAPP architecture, discussing different levels of proxies and related work. Next, section 3 works out a classification scheme for proxy objects on which their management is based. The details of the management of proxies as part of the RAPP architecture are described in section 4. Finally, section 5 summarizes this paper and gives an outlook on future work.

2. The Rapp Architecture

2.1 Related Work

In the area of mobile computing, proxies can be found on two different levels:-

Protocol Proxies operate on the protocol level. They can interfere on the transport level by introducing indirectness like in *Indirect TCP* [2] or by acting on behalf of the mobile client as in the *Snoop* protocol [3]. More specific proxies work on the base of a modified application protocol like a version of HTTP improved for mobile clients [17] or the *Low Bandwidth X Protocol (LBX)* [10] suitable for X-Windows applications running on a mobile client.

Application Proxies additionally modify application data to reduce the required bandwidth or to pre-compute data. One example, covering the area of CAD and network management using a pen-based palmtop computer, is described in [5].

Some approaches combine the two levels of proxies, leading to proxy instances improving the behaviour of protocols and modifying (e.g. compressing or filtering) application data to reduce the required bandwidth. Some examples of this are the *GLOMOP* project [6], its follow-on project *Pythia* [8], the use of *service proxies* [11] and an architecture based on real-time distillation [24]. Other approaches do not only include proxies but define a complete platform for supporting adaptive mobile applications. One of the most interesting approaches is the *Mobiware Toolkit* [4], which provides a set of open programmable interfaces and algorithms and, hence, deals with mobile devices, base stations and mobile-capable switches or routers. However, this architecture demands serious modifications in the distributed applications.

Generally, the following issues have not been solved completely in these approaches:-

- **Extensibility**. Most approaches have developed specific proxies for certain applications or for special data types, but it is often hard to extend these approaches to deal with new applications and data types.
- **Flexible Behaviour**. In most systems the behaviour of the proxy is statically defined. In addition to this, it is generally not possible to fine-tune the proxy's operation to cope with variations in the wireless link's QoS.
- **Flexible Placement**. The location of the proxy is an important factor in determining the performance of the overall proxy system.
- **Scalability**. Finally, there is the issue of scalability. All the described approaches work well, if you have one or two mobile hosts, each being represented by their own proxy, or if you scale to the number of mobile nodes by installing a high-performance computer or cluster of computers to host the proxies (e.g. [7]). However, for the system to scale proxies must be distributed throughout the system.

We have attempted to address the above issues in our architecture, which is described in the next section.

2.2 An Overview of the RAPP Architecture

The RAPP (**R**eactive **A**daptive **P**roxy **P**lacement) architecture is based on the *Common Object Request Broker Architecture (CORBA)* standard [16]. Using CORBA, it is easy to extend existing applications and proxy objects to become part of the RAPP architecture. Proxy objects can be installed on the fly using CORBA object calls. Furthermore, the CORBA adapters for the application components exchange QoS messages via a generic QoS service on which decisions to install or deinstall proxy objects are based. Figure 2 illustrates the RAPP architecture.

One key issue of the RAPP architecture is the management of proxy objects. Therefore, a consistent model for proxy objects was created based on a comprehensive classification of proxy objects, which is detailed in the next section.

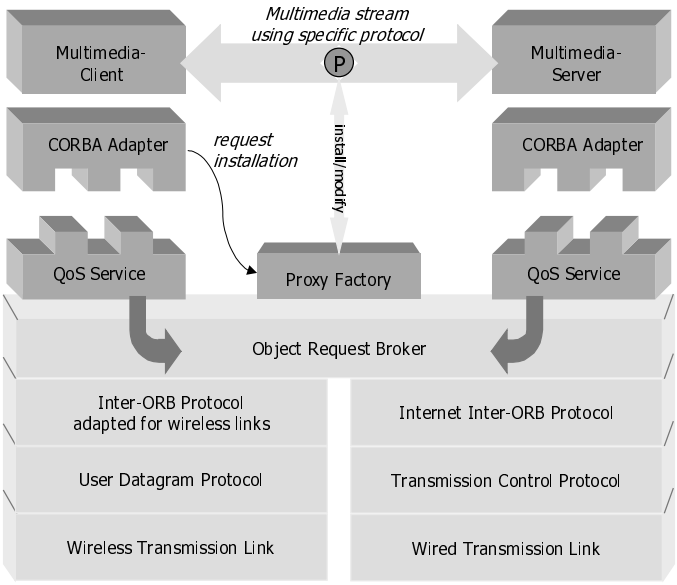

Figure 2: An Overview of the RAPP Architecture

3. Proxy Objects

3.1 Classification of Proxy Objects

One of the goals of the RAPP architecture is to transparently provide proxy objects for distributed applications. This enables a suitable proxy object to be chosen and installed, whenever the actual link's characteristic demands a proxy object. As the collection of different proxy objects offered via the RAPP architecture is independent of the applications, one cannot expect the distributed application to be prepared to cope with any of these proxy objects. However, the application can use the advantages of the proxy objects via generic interfaces. To determine these generic interfaces proxy objects require classification. We propose a hierarchical classification in order to allow flexibility in defining and using these interfaces.

At top level, three main characteristics define the classification scheme:-

1. The proxy object modifies the input stream without changing the stream type. Hence, the types of input and output stream are the same, but the amount of data in the output stream has been changed by (usually loss-prone) compression. This characteristic is called *Data Filtering*.
2. The proxy object receives an input stream, processes the data and sends the processed data in an output stream. This data processing might not only change the amount of data to be sent but also its type. We call this characteristic *Data Type Modification*.
3. Besides processing the data, the proxy object might also be capable of storing the data so that the client can retrieve them later, which we call *Data Caching*.

According to these three main characteristics, three different classes of proxy objects have been identified and are described below:-

Filtering Proxies

A proxy object that cannot cache data and cannot change the data type when processing the incoming stream is called a *filtering proxy*. One example of this is a proxy object acting as an MPEG filter to reduce the required bandwidth by dropping colours or less relevant video frames [22].

Transforming Proxies

A proxy object that modifies the stream type without being able to store the transformed data locally is classified as a *transforming proxy*. Such a proxy might compress an ASCII stream into a compressed binary stream to save bandwidth.

Caching Proxies

In mobile computing the user would sometimes like to work without being connected to the wired network. Therefore, a *caching proxy*, like a mail proxy, might store data for the mobile client so that they can request the stored data after having established a reliable connection to the wired network again.

Furthermore, these proxy classes can also be combined. A transforming proxy might use a loss-prone compression technique, hence it also acts as a filtering proxy, whereas a caching proxy might additionally implement a function to change the type of the stream or to compress data. Because each proxy class results in a specific interface (see section 3.3), a proxy implementing different proxy classes offers the combination of their interfaces.

3.2 Types of Proxy Objects

The next level of the classification scheme is based on the types of streams a proxy object receives and emits. Therefore, we introduce a three-fold stream classification scheme involving a coarse-grain classification, a fine-grain classification and an extension of the latter. In the coarse-grain classification we differentiate between five *stream types* according to the *Multipurpose Internet Mail Extensions MIME* standard [9]. To formally define the stream type we use the Object Type Macro as defined in [18] shown below.

```
streamType OBJECT-TYPE
    SYNTAX  Integer{
               text(1),
               application(2),
               image(3),
               audio(4),
               video(5)}
    ACCESS  read-write
    STATUS  mandatory
    DESCRIPTION
          "To classify a communication data stream coarsely,
           five MIME Content-Types can be used:
             1 – text;
             2 – application;
             3 – image;
             4 – audio;
             5 – video."
    ::= { commDataStreamEntry 2 }
```

Each of the five stream types can be further subdivided using fine-grain subtypes shown in figure 3.

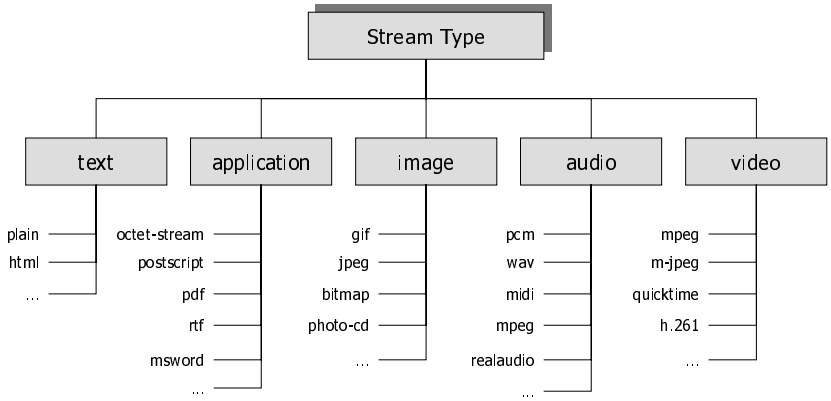

Figure 3: Stream Types and Subtypes

Each of the subtypes can be enhanced by a specific characteristic: examples are "encrypted" or "compressed". To be as flexible as possible both subtypes and subtype enhancements are easily extensible. Any proxy object might then be characterized by the stream it receives and the stream it sends. This characterization is mirrored in the proxy selection process described in section 4.

3.3 Interfaces of Proxy Objects

Each proxy object offers an interface for requesting the different services which the object has implemented. Because the RAPP architecture is a generic architecture, the applications using the services of a proxy object should not be specially customized for these services. Hence, the proxy objects should supply a generic interface making it easier for applications to request services independently of the proxy object's implementation. However, because the proxy objects have different tasks and serve different purposes, we propose to use four levels of interfaces.

Generic Proxy Object Interface

Any proxy object should offer an interface independently of the implemented services. This generic interface allows the following operations:-

- `ProxyObject installProxy (IPAddress sender, integer port, IPAddress receiver, integer recPort, integer StreamID)`
  To install a proxy object, this generic "`installProxy`" operation is used. When calling this operation, one must supply the sender of the stream to be modified by the proxy, the port the stream should be re-routed to, the receiver and its port, and the stream identification. As a result, a handle to the installed proxy object is returned. (Note that this interface procedure is indirectly offered by the proxy factory described in section 4.1.)
- `boolean reRouteProxy (IPAddress sender, integer port, IPAddress receiver, integer recPort, integer StreamID)`
  To modify the route through a proxy, e.g. if a new proxy has to be inserted into the stream or if another proxy has been deleted, the "`reRouteProxy`" proce-

dure is used. It requires the same parameters as the "`installProxy`" procedure, but is directly called at the proxy object's interface.

- `boolean deleteProxy (ProxyObject proxy)`
  A proxy object can be deleted using this operation. Note that the stream has to be re-routed first before deleting the proxy object.

Class-dependent Proxy Object Interface

To influence the behaviour of a proxy object in a general way, the proxy object should offer an interface according to its class. We suggest the following operations: The behaviour of a *filtering proxy* can be controlled using the operations `filterOn()` to enable the filter, `filterOff()` to disable the filter, `increaseFiltering()` to reduce the required bandwidth of the data stream leaving the proxy object and `decreaseFiltering()` to loosen the degree of filtering. A *transforming proxy* offers the operations `transformOn()` to start the transforming process and `transformOff()` to stop it. The user can affect the behaviour of a *caching proxy* using the operations `cacheOn()` to enable the caching, `cacheOff()` to disable caching and `emptyCache()` to delete all the data stored in the cache.

As stated above, a proxy that implements more than one proxy object class must supply an interface combining all the interface operations defined for these proxy object classes.

Type-dependent Proxy Object Interface

Due to the different types of streams a proxy object can process, type dependent operations have to be gathered in a special interface. Because the number of different stream types and their subtypes is large, not all type-dependent interfaces can be detailed here. The following example for a filtering proxy illustrates what a type dependent interface might look like.

In the first prototype implementing the RAPP architecture, we experimented with a proxy object that filters an MPEG video stream [22]. Therefore, this proxy offers a class-dependent interface for filtering proxies consisting of the operations `filterOn()`, `filterOff()`, `encreaseFiltering()` and `decreaseFiltering()`. Furthermore, as a proxy specifically developed to filter an MPEG stream, it offers the following type-dependent operations (amongst others):-

- `dropBFrames()`: erase all the B-frames in the MPEG stream;
- `dropBandPFrames()`: erase all B- and P-frames in the MPEG stream;
- `dropColor()`: change the colors to black & white;
- `requantize()`: requantize the resolution.

All proxy objects suitable for an MPEG stream must implement these interface operations. In addition, these operations may also be used when realizing the operations of the class-dependent proxy object interface: the `filterOn()` operation might be mapped onto the `dropBFrames()` operation, whereas the `encreaseFiltering()` operation would then be performed by calling the `dropBandPFrames()` operation.

Implementation-dependent Proxy Object Interface

Finally, any proxy object implementation might offer an implementation-specific interface. The operations contained in this interface are not restricted in any way.

However, the application must be aware of the implementation if it wants to use any of these operations.

3.4 Location of Proxy Objects

Another critical issue is the location of a proxy object. There are two restrictions that have to be considered when installing a proxy:-

1. The proxy object must be installed on a node where (a) the required resources (hardware, software, qualitative requirements) can be fulfilled and (b) the object *may* be installed.
2. The installation of one proxy object may result in the installation of another proxy object. E.g., if a transforming proxy is installed which compresses data, then another one has to be installed to uncompress the data for the receiver.

The location of a proxy has not been deeply considered in other research projects in this area. The location of the proxy objects might be crucial to the performance experienced by the mobile user, especially for applications distributed over a wide area network. However, due to complex network structures and different network domains, it is generally very complicated to compute the optimal location for proxy installation. Therefore, we propose a simplified decision scheme for proxy placement. For each proxy object, a list of five placement options determines the sequence of location preferences. The five options are [20]:-

1. The proxy should be located on the server.
2. The proxy should be located within the server's domain (e.g., the same subnet).
3. The proxy should be located in the client's domain.
4. The proxy should be located on the client.
5. The proxy could be located anywhere in the network.

For example, the location preference for the MPEG proxy described above is the sequence <1,2,3,5>. This means that the proxy should be installed on the MPEG server if possible to filter the MPEG stream before it leaves the sender. The second option is to install the proxy in the server's domain to minimize the bandwidth needed between the server's subnet and the client. The third option is to install the proxy within the mobile client's actual domain, but preferably within the wired network, to disburden the (possible) wireless link to the mobile user of a stream requesting too much bandwidth. Finally, if all these options are not feasible (due to security or performance reasons), the proxy might be installed somewhere in the network (possibly resulting in a longer transmission delay). However, to install the proxy on the (mobile) client is not advisable, because this would have no effect on the quality of the transmission over the wireless link.

The location preferences play an important role in the management of proxy objects as described in the next section.

4. Management of Proxy Objects

After the classification of proxy objects, this section deals with their management. Therefore, we differentiate between the following tasks:-

- Proxy objects must be hosted for instantiation.
- Suitable proxy objects must be selected.

- Instantiated proxy objects must be parametrised, modified and de-installed.

The following sections describe the components cooperating for the given tasks.

4.1 Proxy Factory

Proxy objects are kept in proxy repositories managed by *proxy factories*. Each proxy factory is responsible for a number of different proxy objects. One can add any proxy object to a given proxy factory by supplying a description of the proxy object. This description contains the following information:

- a classification of the stream received by the proxy object, represented by the triple <streamType, streamSubType, streamSubTypeSpecific>;
- a classification of the stream emitted by the proxy object using the same structure as introduced above;
- the location preference of the proxy object given as a vector containing between one and five integers in the numeric range of one to five, as described in section 3.4;
- a boolean variable specifying, whether the proxy object uses loss-prone compression or not.

Hence, a proxy object that filters an MPEG video stream can be described by the following information:

```
<video, mpeg, none> // Instream is an MPEG video
                    // stream without special characteristics
<video, mpeg, none> // Outstream is an MPEG video stream
                    // without special characteristics
[1,2,3,5]           // Location preferences
true                // The proxy uses loss-prone compression
```

The proxy factory registers all its proxy objects with a local proxy trader. This proxy trading service is detailed in the next section.

4.2 Proxy Trading Service

During the proxy selection phase, the client requests a proxy object to be installed for a given data stream. As described in the section above, the effect of a proxy depends on the place it is installed. In order to simplify the process of finding a suitable proxy object, we suggest the use of a *proxy trading service*. This service is provided by several *proxy traders*. Each proxy trader is responsible for a certain network domain, e.g. a subnet. Via a *registration* process, a proxy trader receives the information about all the proxies available through all the proxy factories of this domain.

In order to register a proxy object, the proxy factory responsible for that object has to export its service description to the proxy trading service. Following [1], this "`export`" procedure must supply an object reference to the object being offered, i.e. the proxy factory's ID, a service type name and a set of properties, given as name and value pairs. In the case of the proxy trading service, the service type name is the class of the proxy object (as defined in section 3.1) and the set of properties is given through the information on the proxy object given above. As a result, the trader returns a proxy offer ID, which is used for further modifications of the registered service, including its withdrawal.

To de-register a proxy object, the "`withdraw`" procedure must be called supplying the proxy offer ID. Furthermore, it is sometimes advantageous to have a procedure de-registering all the proxy objects of a given proxy factory. For this, the CORBA standard suggests the "`withdraw_using_constraints`" procedure.

Finally, the "`modify`" procedure can be used to modify an already registered proxy offer specifying its offer ID and a new set of properties.

To keep the proxy trading service simple, the standardised trader operations "`describe`" and "`resolve`" are not currently included.

Since each domain has its own proxy trader, it is easy to assign a proxy trader to both servers and mobile clients. When the client first uses a service offered by the server, it also receives a handle to the server's proxy trader. This is important, because some proxy objects might prefer to be installed close to the server.

Once the client decides that a proxy object should be installed into the stream, it addresses the proxy trading service using the "`query`" procedure supplying the following parameters:-

- the client's ID;
- a reference to the proxy trading service of the server's domain;
- a description of the desired proxy service (according to the proxy object description above, but without location specification);
- a preference scheme to influence the sequence of results;
- an integer value ("`how_many`") for describing the maximum length for the sequence of results.

This query leads to the proxy trading service trying to find a sequence of proxy factories that offer proxy objects of the required kind via the following three steps (see figure 4):-

1. The trading service searches its local database where the proxy offer descriptions are stored. All the offers that allow local installation and have suitable input and output streams are gathered in a vector V1. Furthermore, the trader creates a vector V2 of all the offers whose outgoing streams correspond to the one the client expects. These offers might be combined with a proxy object to be installed on the server side. Hence, the trader will send additional queries to find complement proxies to the ones stored in V2.
2. After the proxy trading service of the server's domain has received the original query it performs the retrieving of its local database, but without looking for offers fitting the output stream only. The same is done for the additional queries looking for complementary proxy objects. For each query a result containing the offers and the corresponding proxy factories is returned.
3. The client's proxy trading service then merges the results of its local query stored in V1 and the first result returned by the server's proxy trading in vector V3. Then it combines the offers stored in vector V2 with the results of the additional queries and extends vector V3 with the suitable proxy object pairs. If the merged list is empty, the proxy trading service could then query other proxy trading service instances, but only if the client did not prevent that in its "`preference`" parameter.

If the result list is not empty, the proxy trading service orders the entries according to the "`preference`" parameter given in the query. It also considers the "`how_many`" parameter for only returning the best *x* results to the query. It is then up to the client to decide which of the offers included in the returned list to choose.

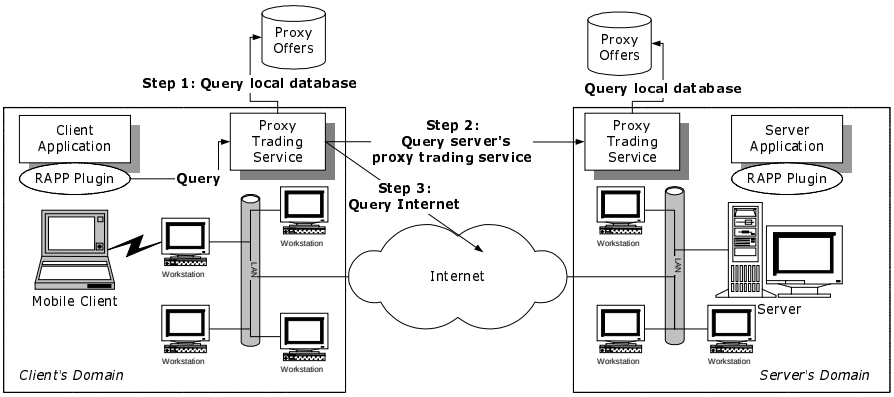

Figure 4: Functionality of the Proxy Trading Service

4.3 Proxy Installation, Modification and De-installation

Having received the results to its query, the client chooses the proxy factory (or factories, if there are two proxy objects to be installed) and requests the installation of the proxy object. For the installation, the procedure "`installProxy`" is called, which is part of the generic proxy object interface and is offered indirectly by the proxy factory.

The client can then enable the functionality of the proxy object by calling the functions of the class-dependent proxy interface. Based on monitoring the actual QoS associated with the stream, the client is able to modify the behaviour of a proxy object, choosing from three options:-

1. The class-dependent proxy interface offers high-level operations to influence the functionality of the proxy object.
2. A finer degree of control is offered through the type-dependent proxy interface. However, the client's application must be prepared to use these operations.
3. Finally, the client could use the implementation-dependent interface offering low-level operations.

Once the actual QoS is sufficient or the data stream has finished, a proxy object can be de-installed using the "`deleteProxy`" operation. Note that de-installing a proxy object might result in having to re-route the stream calling the "`reRouteProxy`" operation of other installed proxy objects.

4.4 The RAPP Prototype

For a first prototype we chose the following components:

- A modification of the Berkley MPEG player is used as a client/server application [19].

- As a proxy object, we used an MPEG filter developed by N. Yeadon [23].

We had to enhance these components with special CORBA stubs (as can be seen in figure 5, called "RAPP Plugin" consisting of the generic QoS Service component and the CORBA adapter illustrated in figure 2). These enhancements were programmed using the OmniBroker as an Object Request Broker implementation [13]. The video stream emitted by the MPEG video player is sent using UDP packets. These UDP packets are not touched by the RAPP plugins. The plugins are only responsible for signalling.

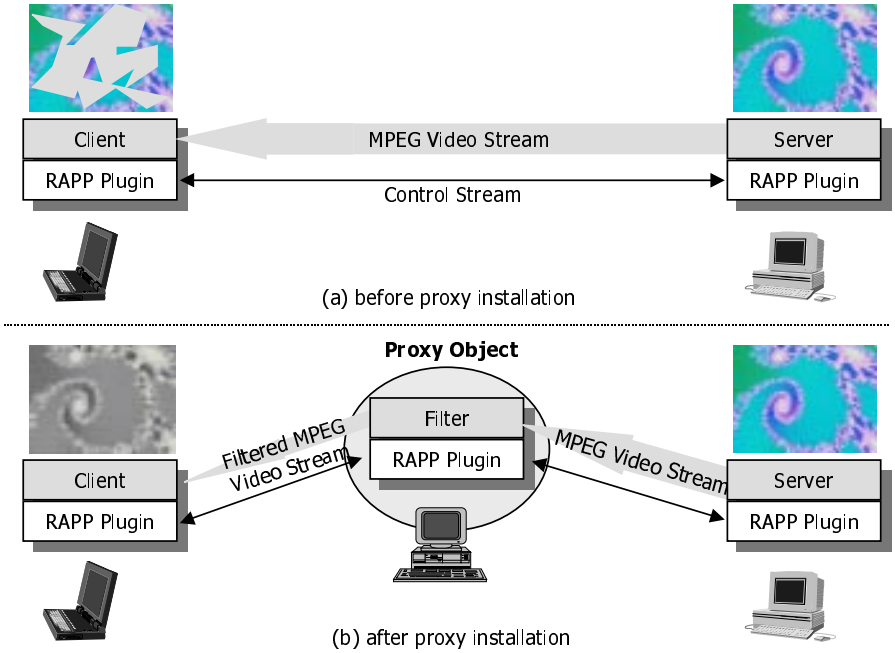

Figure 5: Functionality of the Prototype

We organized the scenario as follows:

- The client part was started on a Linux Pentium-based PC in the Computing Department of Lancaster University.
- The server part ran on another Linux Pentium-based PC in the same department.
- The proxy trader service was supplied by a third PC in the Computing Department.

The MPEG filter we chose as a proxy was, however, installed on a Sun Workstation in the Institute of Telematics in the University of Karlsruhe/Germany. This meant, that the multimedia stream containing the MPEG video frames had to be rerouted over Karlsruhe.

The results of this experiment have been very satisfying. Installing and deinstalling the proxy worked without any kind of disturbing delay. Although the application parts and the MPEG filter used as a proxy object ran on different ma-

chine architectures and operating systems, CORBA provided a seamless integration of the different application parts. This prototype is a solid foundation to extend the implementation of the RAPP architecture.

5. Conclusion

Proxies have been proven to be an effective means of adapting the QoS requirements of multimedia applications to the varying characteristics of wireless links. In order to be flexible when supporting different kinds of distributed applications and multimedia streams we introduced an architecture for managing proxy objects based on the CORBA standard.

Key factors for the success of this architecture are a comprehensive technique to express the characteristics of proxies for the proxy selection process and an easy to handle interface to interact with proxy objects. This interface makes it easy to call operations and, thus, change the proxy's behaviour, without having to know the proxy's functionality in detail.

The prototype of the architecture has been implemented and tested both locally and over wide distances (between England and Germany). The results have been very promising, and therefore we plan to enhance the prototype in order to produce performance data. Furthermore, because installing and modifying proxies is critical for security, the RAPP architecture will also be extended to contain a security mechanism to control the handling of proxies.

Acknowledgements

This work has been partly supported by the EPSRC under the auspices of the Reactive Services project.

References

[1] S. Baker: *CORBA Distributed Objects Using Orbix*. ACM Press / Addison Wesley, Harlow, England - Reading, Massachusetts, 1997.

[2] A. Bakre and B.R. Badrinath: *I-TCP: Indirect TCP for Mobile Hosts*. 15th International Conference on Distributed Computing Systems, Vancouver, Canada, May 30 - June 2, 1995.

[3] H. Balakrishnan, S. Seshan and R.H. Katz: *Improving Reliable Transport and Handoff Performance in Cellular Wireless Networks*. ACM Wireless Networks 1(IV), 1995.

[4] A.T. Campbell: *The Mobiware Toolkit*, Fourth International Workshop on High Performance Protocol Architectures (HIPPARCH '98), London, June 15–16, 1998

[5] W. Citrin, P. Hamill, M.D. Gross and A. Warmack: *Support for mobile Pen-based Applications*. Third annual ACM/IEEE International Conference on Mobile Computing and Networking MobiCom'97, Budapest, Hungary, September 26–30, 1997.

[6] A. Fox and E.A. Brewer: *GloMop: Global Mobile Computing by Proxy*. http://www.research.microsoft.com/os/sosp%2D15/fox.txt, 1995.

[7] A. Fox and E.A. Brewer: *Reducing WWW Latency and Bandwidth Requirements by Real-Time Distillation*. Fifth International World Wide Web Conference, Paris, France, May 6-10, 1996.
[8] A. Fox, S.D. Gribble, E.A. Brewer and E. Amir: *Adapting to Network and Client Variability via On-Demand Dynamic Distillation.* ASPLOS-VII - Seventh International Conference on Architectural Support for Programming Languages and Operating Systems, Cambridge, Mass., USA, October 1–5, 1996.
[9] N. Freed and N. Borenstein: *Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types*. Request for Comments 2046, November 1996.
[10] J. Fulton and C. Kantarjiev: *An Update on Low Bandwidth X (LBX)*. The X Resource 5(1), pp. 251–266, January 1993.
[11] A. Hokimoto, K. Kurihara and T. Nakajima: *An Approach for Constructive Mobile Applications Using Service Proxies*. 16th International Conference on Distributed Computing Systems ICDCS, Hong Kong, May 27–30, 1996.
[12] R.H. Katz: *Adaptation and Mobility in Wireless Information Systems*. Personal Communications 1(1), pp. 6–17, First Quarter, 1994.
[13] M. Laukien and U. Seimet: *OmniBroker*, Object-Oriented Concepts, Inc., Billerica (USA) and Ettlingen (Germany). Manual, December 16, 1997.
[14] S.W. Lodin and C.L. Schuba: *Firewalls fend off invasions from the Net*. IEEE Spectrum 35(2), pp. 26–34, February 1998.
[15] A. Luotonen and K. Altis: *World-Wide Web Proxies*. First International Conference on the World-Wide Web, Geneva, Switzerland, May 25–27, 1994.
[16] OMG: *The Common Object Request Broker: Architecture and Specification*, Object Management Group (OMG). Revision 2.2, February 1998.
[17] V.N. Padmanabhan and J.C. Mogul: *Improving HTTP Latency*. Computer Networks and ISDN Systems 28(1/2), pp. 25–35, December 1995.
[18] M.T. Rose and K. McCloghrie: *Concise MIB Definitions*. Request for Comments 1212, March 1991.
[19] L. Rowe, S. Smoot and E. Hung: *MPEG Research at U.C. Berkeley*. http://bmrc.berkeley.edu/projects/mpeg/, 1998.
[20] J. Seitz, N. Davies, M. Ebner and A. Friday: *A CORBA-based Proxy Architecture for Mobile Multimedia Applications*. Second IFIP/IEEE International Conference on Management of Multimedia Networks and Services MMNS'98, Versailles, France, November 16–18, 1998.
[21] T. Urquhart: *Network and Service Management - Why CORBA?* NOC'98 - Third European Conference on Networks and Optical Communications, Manchester, UK, June 23–25, 1998.
[22] N.J. Yeadon, F. García, D. Hutchison and D. Shepherd: *Filters: QoS Support Mechanisms for Multipeer Communications*. IEEE Journal on Selected Areas in Communications 14(7), pp. 1245–1262, September 1996.
[23] N.J. Yeadon: *Quality of Service Filtering for Multimedia Communication.* PhD-thesis. Computing Dept., Lancaster University, Lancaster, UK, 1996.
[24] B. Zenel and D. Duchamp: *A General Purpose Proxy Filtering Mechanism Applied to the Mobile Environment*. Third annual ACM/IEEE International Conference on Mobile Computing and Networking MobiCom'97, Budapest, Hungary, September 26–30, 1997.