Offensive and Defensive Adaptation in Distributed Multimedia Systems♣

Roland Tusch [1]*, László Böszörményi [1], Balázs Goldschmidt [2],
Hermann Hellwagner [1], Peter Schojer [1]

[1] Institute of Information Technology, Klagenfurt University, Klagenfurt, Austria
[2] Department of Control Engineering and Information Technology, Budapest University of Technology and Economics, Budapest, Hungary

Abstract. Adaptation in multimedia systems is usually restricted to defensive, reactive media adaptation (often called stream-level adaptation). We argue that offensive, proactive, system-level adaptation deserves not less attention. If a distributed multimedia system cares for overall, end-to-end quality of service then it should provide a meaningful combination of both.
We introduce an adaptive multimedia server (ADMS) and a supporting middleware which implement offensive adaptation based on a lean, flexible architecture. The measured costs and benefits of the offensive adaptation process are presented.
We introduce an intelligent video proxy (QBIX), which implements defensive adaptation. The cost/benefit measurements of QBIX are presented elsewhere [1].
We show the benefits of the integration of QBIX in ADMS. Offensive adaptation is used to find an optimal, user-friendly configuration dynamically for ADMS, and defensive adaptation is added to take usage environment (network and terminal) constraints into account.

Key words: stream-level adaptation, server-level adaptation, MPEG-4, MPEG-7, MPEG-21.

♣ This research project is funded in part by FWF (Fonds zur Förderung der wissenschaftlichen Forschung), under the projects P14788 and P14789, and KWF (Kärntner Wirtschaftsförderungsfonds).

* Corresponding author. Tel.: +43/463/27003622. Fax: +43/463/27003699
Email addresses: roland@itec.uni-klu.ac.at (Roland Tusch), laszlo@itec.uni-klu.ac.at (László Böszörményi), balage@inf.bme.hu (Balázs Goldschmidt), hellwagn@itec.uni-klu.ac.at (Hermann Hellwagner), pschojer@itec.uni-klu.ac.at (Peter Schojer).

1. Introduction

Adaptation is - in a general sense - the capability to respond to changes of the environment by the change of some own characteristics, without losing the own identity. If a living creature feels hungry, but sufficient food is not available, then it has two choices for survival: Either it is able to enlarge the available food resources, e.g. by moving to a better place, or to change its inner need and learn to be satisfied with less or worse food. The first approach could be called *offensive*, the second *defensive* adaptation. Obviously both have their limits and have a common prerequisite: flexibility. (Our creature has to be flexible either to go outside and find new places or to go inside and be satisfied with less food - or to do both, if e.g. a new place can be found, which is, however, still not sufficiently rich of food.) Offensive adaptation is usually *proactive* (it is better to start to look for an opulent place before getting too hungry), whereas defensive adaptation is usually *reactive* (if there is no food, there is no other choice than getting humble).

In distributed multimedia systems - especially in video management systems - adaptation is becoming increasingly important. The reason is simple: due to the challenging amount of data involved and the soft real-time constraints, video management systems are always ”resource hungry”. This hunger can be satisfied neither by best-effort resource management, which cannot handle timing constraints, nor by reservation, which over-allocates resources (for the worst case). Therefore, on the long term, adaptation is a necessity. It is - fortunately - also an opportunity, both in its defensive and in its offensive forms.

To find new places and new food, we have to be able to *communicate* with the environment, we must understand its signals. In a computerized world this means: being able to support *standardized* communication.

Fairly much research has been done on *media adaptation*, which is obviously defensive [2–5]. The required flexibility lies in the inner structure of the media data. Especially by sophisticated video coding techniques, ample space is created for transcoding of video data resulting in smaller size and bitrate and in still acceptable perceptual quality. Fortunately, the relation between size and quality reduction is usually non-linear: up to a certain limit, large size reduction causes a moderate quality loss.

Much less research has been done on offensive (also called *server-level*) adaptation. The reason for this reduced interest lies probably in the fact that most distributed multimedia systems lack the necessary flexibility. Such a system must be able to dynamically acquire new resources and move/replicate its own code and data as needed. This requires flexibility of the architecture and consequent adherence to standards – both rare characteristics in current distributed multimedia systems. In the course of the ADMITS (Adaptation in Distributed Multimedia IT Systems) project [6,7] we address both kinds of adaptation and their integration as well.

Offensive adaptation is realized by an Adaptive Distributed Multimedia Server (ADMS), which is able to allocate new server nodes on the network and migrate functionality and data to them on demand. ADMS has a highly flexible architecture enabling this kind of migration operations. If for example the *host*

recommender component notices that a certain group of clients could be better served if the data collector component (collecting stripe units from the data storage nodes and streaming them to the clients) were physically located near to this client group, then it simply allocates a new node and sends a copy of the data collector code there. A detailed description of ADMS is given in Section 3.

Defensive adaptation is realized by a Quality Based Intelligent Proxy (QBIX). The proxy is able to handle different quality levels of the same video (based on MPEG-4 [8] coding and on transcoding, supported by MPEG-7 [9] meta data). It can operate as a cache, in which case it uses adaptive cache replacement algorithms. Instead of discarding replacement candidates it reduces their quality as long as possible, thus raising the hit rate considerably. It can also operate as a gateway, in which case it serves client devices with different capabilities by videos of different quality. It can e.g. send a high quality version of a video to clients on a personal computer and a low quality version to clients with a PDA. A detailed description of QBIX is given in Section 4.

An integration of the offensive and defensive adaptation is given if we use the intelligent proxy itself as a component of the adaptive server. Due to the flexible architecture this is technically easy. The usual strategy is that we first try to apply offensive adaptation by enlarging the server by new nodes to host the proxy functionality. As the proxy is able to perform defensive adaptation, it still can reduce the quality of the stored videos if necessary. Whereas ADMS and QBIX are already fully operational, the integration is still in progress. The quantitative evaluation of the effect of the integration is the next step in the ADMITS project.

2. Related Work

In [10] it is shown that distributed multimedia servers have benefits over single server architectures regarding scalability and server-level fault tolerance. We discussed in earlier papers [11,12] that existing distributed server architectures like the Berkeley Distributed VoD System [13], the Tiger Video Fileserver [14], or the EURECOM VoD Server [15] have a monolithic architecture and are performance-optimized to one main goal: serving thousands of simultaneous client requests. However, in heterogeneous environments, it is usually not the server, but the network that becomes a bottleneck. This is especially the case if a certain level of quality of service is to be guaranteed in terms of latency times, bandwidth availability or packet loss.

Although sophisticated commercial systems like the **`Darwin Streaming Server`** (DSS) – an open source version of Apple's modular **`QuickTime Streaming Server`** – or the **`Helix Universal Server`** (HUS) [16,17] – developed by RealNetworks –, show a highly distributed architecture, their organization is static, they cannot acquire new resources on the network on demand.

Content distribution networks (CDNs) are dedicated collections of servers (called **`data centers`**) strategically located across the Internet. A typical

example for a CDN is Akamai's distributed content delivery system, which deploys more than 12000 servers in over 1000 networks around our planet [18]. Based on the assumption that the strategic locations of the data centers are well chosen, a CDN is definitively a good solution for adapting the physical location of a media stream. It is highly available and serves the streams from locations where clients perceive a good quality of service. However, the strategic positioning of data centers and edge servers is typically done manually by observing client demands. A CDN does not provide means for performing strategic placements automatically.

In a peer-to-peer (P2P) file sharing network, peers collaborate to form a distributed system for the purpose of exchanging content [19]. However, the applicability of P2P networks for adapting the location of media streams in a distributed streaming environment is rather poor. In particular, it faces the following two problems: (i) Participation in a P2P network is purely voluntary. A recent study has shown that most peers are run by end users, who suffer from low availability, and have network connections with a relatively low capacity [20]. (ii) It is server-less, which makes a controlled distribution of content very difficult [21]. Nevertheless, there are approaches that use P2P content delivery for media streaming, as e.g. presented in [21] and [22].

An offensive server architecture requires a QoS-aware middleware providing active support for adaptation steps. Extensive work has been done in the area of QoS-aware middleware for ATM-based networks and also for systems using RSVP [23–25]. These systems are hardly appropriate for the Internet, where resource reservation is still rather theory than practice. On the other hand, considerable work has been done on end-to-end distance monitoring and estimation in the Internet [26–29]. Such bandwidth and delay measurement/estimation algorithms can be used to approximate QoS awareness for a middleware as needed for offensive adaptation. Although certain middleware systems support dynamic replication or migration of services and components, like Jini [30] or Symphony [31], they do not provide measurements and estimations of network distances and server resources.

One possibility to cope with typical Internet problems such as network jitter and bandwidth limitations is media adaptation. QBIX [1] is a quality based intelligent proxy that caches whole videos and offers media adaptation support to reduce the size/bandwidth requirements of the cached videos. Related work in this area is sparse. Examples are periodic caching of layered coded videos [32], combination of replacement strategies and layered coded videos [33], quality adjusted caching of GoPs (group of pictures) [34], adaptive caching of layered coded videos in combination with congestion control [35] or simple replacement strategies (patterns) for videos consisting of different quality steps [36]. Most of these proposals rely on simulation to evaluate the performance of the caching techniques. Therefore some assumptions have to be made about the structure of the videos (e.g. layered videos).

Because QBIX supports re-encoding of videos without layered encoding, a real implementation was used to evaluate the impact of re-encoding on the quality of cache replacement, something which is hard to simulate. The only other project we know about, also offering a real implementation is described in

[35]. This work relies on proprietary systems and protocols, whereas QBIX integrates modern multimedia standards like MPEG-4 [8], MPEG-7 [9], the upcoming MPEG-21 standard [37] and communication standards like RTP and RTSP [38]. Integrating QBIX into ADMS will create the very first distributed multimedia server that combines offensive and defensive adaptation.

3. QoS-driven Server-level Adaptation

In a statically configured distributed server architecture with a behavior steered by several sorts of distance metrics to the clients (e.g. by evaluating RTCP statistics of former clients), client requests can either be accepted or denied (admission control). There is no third option. In cases of increased denials due to network resource shortages, such a server has no chance to adapt its layout to serve more clients. Defensive media adaptation (see more in Section 4) cannot always cope with such situations either, because (1) adaptation of the media content is not always possible and (2) it may even be prohibited by the content provider.

In the sequel, we therefore introduce an *adaptive* distributed server architecture that can modify its configuration according to client requests. We discuss this architecture and evaluate its performance behavior.

3.1. Components of a Distributed Multimedia Streaming Server

3.1.1. Guidelines for Component Identification

In [11,12] we describe the architecture of an adaptive distributed multimedia streaming server (ADMS), which explicitly controls its own layout. There is a trade-off between the grade of flexibility and the complexity caused by too large a number of components. We identified four basic components that are necessary and sufficient for the composition of a dynamically reconfigurable streaming server. Figure 1 illustrates a sample combination of these components, including the standard protocols used.

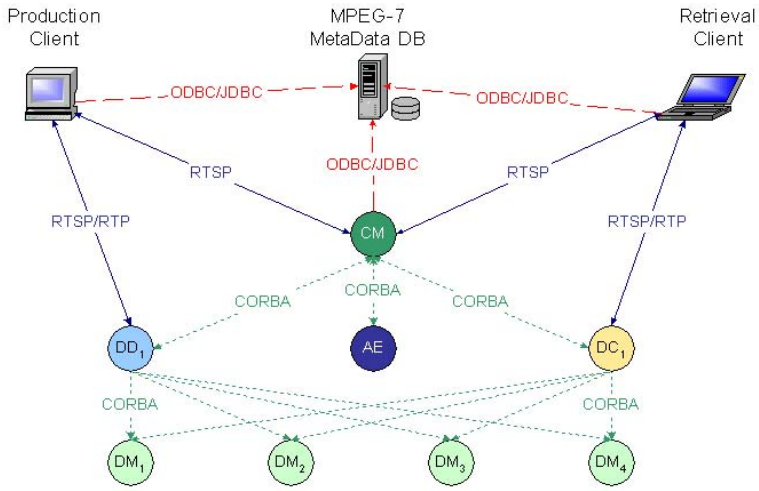


Fig. 1. Components and Protocols Used in a Component-based Streaming Server Environment

The following two guidelines have to be taken into account when designing a component-based distributed streaming server. First, each component must be *independent, reusable, adaptable,* and *combinable* as much as possible. These considerations are valid for a static distributed environment as well. For a dynamically adaptable system the components must additionally be *movable*.

Second, in order to get a lean structure with a strongly limited number of components, each component should fulfill one substantial logical task. Such substantial tasks are data acquisition, data streaming, data storage management and overall control. For example, during a data acquisition scenario initiated by a production client, a media stream has to be distributed (”striped”) among a set of server nodes. One single component is needed to receive the stream data, to split the stream into smaller pieces, and to distribute the pieces among a set of data nodes. The data nodes themselves are equipped with a component for storing and retrieving pieces of media data. Thus, a distribution component can be combined with a number of storage components, which all may run on separate server nodes.

Following these guidelines, four basic components have been identified to constitute a component-based distributed streaming server: *data managers* (DMs), *data distributors* (DDs), *data collectors* (DCs), and *cluster managers* (CMs). The implementation of the components can use different technologies, interoperability is achieved by using CORBA as communication medium [39].

3.1.2. Data Manager

Data managers are the key components in the ADMS architecture. A data manager provides means for efficient storage and retrieval of elementary streams or segments thereof. Since one elementary stream or segment may be striped among a number of data managers, each data manager only stores a

portion of the stream or segment. Figure 2 illustrates the internal storage organization of the media streams. The data manager stores a set of partial media streams, which themselves consist of a set of leaf and compound media segments. Real media data is only stored in leaf media segments. Compound segments are used to describe syntactic and semantic relations among elementary segments.

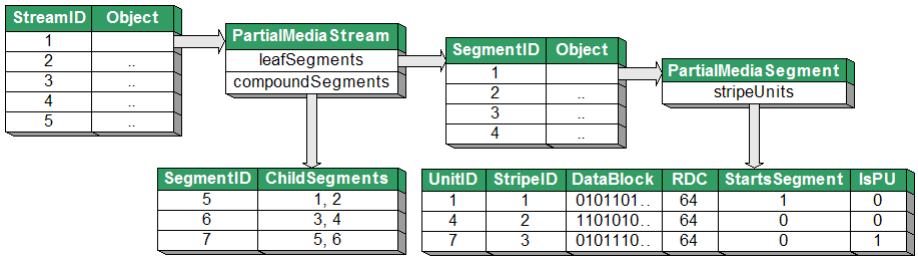


Fig. 2. Objects Comprising a Data Manager Component

3.1.3. Data Distributor

A data distributor component is responsible for the distribution of media data received from a production client or a live camera to a selected set of data managers. The unit of distribution is a so-called *stripe unit*, which can be either of constant data length (CDL), or of constant time length (CTL). In CDL mode, parity units are generated in order to cope with data manager failures. The number and location of target data managers and the mode of striping (single, narrow or wide) is advised by the cluster manager component.

In cases of a non-live source, the process of distribution can be driven by MPEG-7 [9] metadata which describe the temporal decomposition of the media stream. A media stream can be decomposed into a number of segments, organized in an arbitrary number of levels. Figure 3 presents a sample temporal decomposition of a video stream into two segments, using an MPEG-7 descriptor.

The data distributor distributes only elementary streams, since the target system is designed for streaming scenarios based on RTP. Thus, if the media source contains a multiplexed stream, it has to be demultiplexed into elementary streams before striping.

```
<Mpeg7 xmlns="urn:mpeg:mpeg7:schema:2001" xmlns:mpeg7="urn:mpeg:mpeg7:schema:2001"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:mpeg:mpeg7:schema:2001 Mpeg7-2001.xsd">
   <Description xsi:type="ContentEntityType">
      <MultimediaContent xsi:type="VideoType">
         <Video id="1" mediaTimeBase="MediaLocator" mediaTimeUnit="PT1N30F">
            <MediaTime>
               <MediaTimePoint>T00:00:00</MediaTimePoint>
               <MediaDuration>PT10S</MediaDuration>
            </MediaTime>
            <TemporalDecomposition gap="false" overlap="false">
               <VideoSegment id="0">
                  <MediaTime>
                     <MediaRelIncrTimePoint>0</MediaRelIncrTimePoint>
                     <MediaIncrDuration mediaTimeUnit="PT1N30F">99</MediaIncrDuration>
                  </MediaTime>
               </VideoSegment>
               <VideoSegment id="1">
                  <MediaTime>
                     <MediaRelIncrTimePoint>100</MediaRelIncrTimePoint>
                     <MediaIncrDuration mediaTimeUnit="PT1N30F">250</MediaIncrDuration>
                  </MediaTime>
               </VideoSegment>
            </TemporalDecomposition>
         </Video>
      </MultimediaContent>
   </Description>
</Mpeg7>
```

Fig. 3. Temporal Decomposition of a Video Stream using MPEG-7

3.1.4. Data Collector

The data collector performs the inverse operation of data distribution. It collects stripe units of a certain media stream from the appropriate set of data managers, re-sequences the units and sends the buffered stream to the client via an RTP connection. It provides server-level fault tolerance by exploiting parity units in case of unavailable data managers. The collector may also incorporate a caching component, thus reducing client startup latencies and bandwidth consumption. In particular, the data collector can play the role of a proxy server, which is dynamically assigned to serve a group of clients by the cluster manager. This constitutes the difference to usual proxies, which are selected by the clients themselves.

An interesting approach is the integration of gateway functionalities into the data collector. A gateway performs transcoding in order to adapt to given network or client device constraints. It may e.g. reduce the resolution of a video which is sent to a client equipped with a small-resolution screen. A data collector incorporating gateway functionality is a particularly appealing example for the combination of offensive and defensive adaptation. We use offensive adaptation to bring the gateway to the proper place, and there use defensive adaptation to comply with the given client capabilities.

3.1.5. Cluster Manager

A cluster manager is the initial entry point for clients. In contrast to the three component types described before, there is (usually) only one instance of it in an ADMS cluster. It dynamically manages the locations of the other three component instances and maintains knowledge about the distribution of elementary streams among data managers.

An important point is that the cluster manager does not serve client requests by itself. Instead, it redirects a request to an appropriate data collector or distributor, respectively. The most appropriate node to host a data collector or distributor is found by the so-called *adaptation engine* (abbreviated as AE in Figure 1), which is a sub-component of the cluster manager. In a static distributed multimedia server (SDMS) system the adaptation engine may perform several selection algorithms for finding the most appropriate candidate among the given set of nodes. However, if no appropriate candidate can be found, the request has to be denied. In contrast, in an ADMS environment, the adaptation engine may transmit (replicate or migrate) a data collector or distributor instance to an idle server node candidate, thus satisfying requests which might have been denied by a static architecture.

3.1.6. Protocols for Component and Client Interaction

As demonstrated in Figure 1, the only control protocol used between clients and DMS components is the RTSP [38] protocol. Keeping the client implementation CORBA-unaware has the benefit that it may interact with any streaming server which conforms to media streaming standards. The control protocols used internally between DMS components are CORBA-based and conform to the IDL specifications of the components. The ”master mind” behind all kind of scenarios is an MPEG-7 meta database, which stores syntactic and semantic information about the media streams stored on the server.

3.2. ADMS: An Adaptive Distributed Multimedia Streaming Server

The components introduced in Section 3 could be parts of either a *static* or a *dynamic* distributed multimedia streaming server (*SDMS* resp. *ADMS*).

Before introducing the middleware, which is the engine making the server dynamic (see Section 3.2.2), we discuss the main shortcomings of a static architecture.

3.2.1. Discussion of the Static Architecture

In [10] it is claimed that distributed video servers are load balanced regardless of the skew in video popularities. This is only valid for an architecture where the data collectors are integrated into the client application

(*proxy-at-client* model). Consider that there are *M* data manager and *N* data collector instances in an SDMS, supporting the *proxy-at-server* or the *independent-proxy* model. Each server node has the same capacity *C*, and each client request demands the same amount of resources *c* on a data collector (Figure 4).

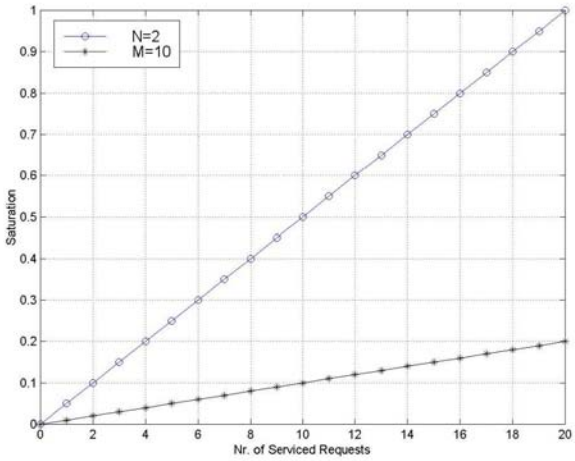


Fig. 4. Host Resource Saturation Progression in an SDMS

In a fair resource reservation scheme with a uniform distribution of requests to the data collectors, N data collectors can admit at most $\left\lfloor \frac{N \cdot C}{c} \right\rfloor$ simultaneous requests. On the other hand, a requested media object is striped among the M data managers, resulting in a capacity demand of $\frac{c}{M}$ on each data manager on average. Thus, the total number of client requests that can be admitted on one data manager node is bound by $\left\lfloor \frac{C \cdot M}{c} \right\rfloor$. Since in a typical static server configuration M is a multiple of N, overall load-balancing is not given anymore. Figure 4 illustrates this imbalance showing that although the M=10 data manager nodes are still under-utilized at 20%, the N=2 data collectors are reaching their saturation point. The assumed resource demand ratio is δ = 0.1 per request on a data collector.

To get rid of the shortcomings of the SDMS architecture, an infrastructure is required which allows for a dynamic composition of the components described in section 3.1, resulting in an adaptive distributed multimedia streaming server (ADMS). Components should be replicable and/or migratable on demand, allowing the dynamic composition of a virtual server [31]. We developed a CORBA-based infrastructure called Vagabond2 [40], which supports dynamic instantiation, migration, replication and evacuation of so-called *adaptive applications* (a synonym for component in Vagabond2).

3.2.2. Vagabond2: A Middleware for Virtual Servers in Internet Settings

Vagabond2 is a CORBA-based middleware, implemented in Java, consisting of two general modules: a module for *component management*, and a module for *component adaptation* (see Figure 5). The component management module provides two services for a distributed server like ADMS: an *application service* for component movement between Vagabond2 hosts, and a *host service* for registering and querying *harbours*. A harbour represents the runtime environment which must be running on each Vagabond2 host. Vagabond2 enables loading the Java byte code of an adaptive application, and instantiating this as a CORBA servant.

The component adaptation module provides two central services for offensive server adaptation: the *adaptation service* and the *resource broker*.

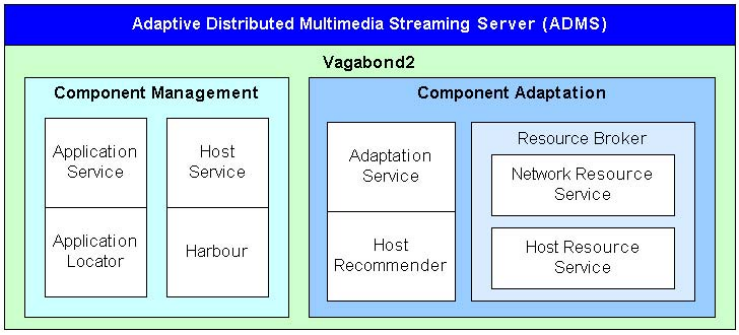


Fig. 5. Modules of the Vagabond2 Middleware

The adaptation service provides the so-called *host recommender*, which tries to find an optimally located host for a certain component under a given set of constraints. An example for a set of QoS constraints described by an MPEG-21 descriptor embedded in an RTSP SETUP message is given in Figure 6.

The MPEG-21 [37] network characteristics descriptor is rooted by the *DIA* (*Digital Item Adaptation*) element. The capability element of the descriptor says that a bandwidth range between 32 and 128 kbit/sec is acceptable, packets might be delivered out of order and may be lost. The condition element specifies that the 128 kbit/sec link may be fully utilized, but on average should only get 64 kbit/sec. The packet delay from a data collector to the client should not be greater than 500 msec, and delay variation not greater than 100 msec. Finally, the packet loss rate must be below two percent.

```
SETUP rtsp://sdms.itec.uni-klu.ac.at/Sample.mp4 RTSP/1.0
CSeq: 2
Range: npt=20-40;time=20030301T140000Z
Content-Type: application/mpeg21_dia
Content-Length: 557
<?xml version="1.0" encoding="UTF-8"?>
<DIA xmlns="urn:mpeg:mpeg21:dia:schema:2003" xmlns:mpeg7="urn:mpeg:mpeg7:schema:2001"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="urn:mpeg:mpeg21:dia:schema:2003 UsageEnvironment.xsd">
   <Description xsi:type="UsageEnvironmentType">
      <Network>
         <Capability maxCapacity="128000" minGuaranteed="32000"
                     inSequenceDelivery="false" errorDelivery="true"/>
         <Condition>
            <Utilization maximum="1.0" average="0.5" avgInterval="1"/>
            <Delay packetOneWay="500" delayVariation="100"/>
            <Error packetLossRate="0.02"/>
         </Condition>
      </Network>
   </Description>
</DIA>
```

Fig. 6. RTSP SETUP Message Including an MPEG-21 DIA Descriptor

The RTSP message further indicates (field *Range*) that the client wants to get the stream segment from the 20th to the 40th second (media relative time). Additionally, the request should be scheduled for March 1st, 2003, at 14 o'clock. This time indication at session setup enables the adaptation engine to proactively schedule the request to an appropriate data collector, taking into account the given QoS constraints, and the location of the data managers, where the stream data has to be retrieved from.

When a new client request *R* arrives, the host recommender has to work out which host should run the data collector instance servicing *R*, and whether it should be an existing or a new (replicated or migrated) instance. The host recommender has to solve the *dynamic server selection* problem to find the best available data collector. If no appropriate data collector exists at all, an SDMS has to reject the request. In contrary, the ADMS may have enough time to create a new place by solving the *capacitated facility location* problem [41].

In order to prepare these decisions, the host recommender cooperates with the *resource broker* - the second service of the adaptation module. The broker provides means to admit immediate and future QoS-constrained requests, and to perform logical reservations of network and host resources. It achieves this by using two additional services: the *network resource service*, and the *host resource services*, the latter running on each Vagabond2 harbour.

An excerpt of the network resource service's IDL specification is shown in Figure 7. Via the *NetworkInfo interface* we can get for example the capacity of a route between two arbitrarily connected Vagabond2 server hosts. Supported *distance metrics* are bandwidth, delay, packet loss rate, and round trip time. In this context, the specification of a period is important. If the period covers future time points, the service provides an estimated value, by applying time series analysis based on measurements. The network resource service requires that on each Vagabond2 harbour a network monitor is running, which periodically

measures instantaneous values of all supported metrics. For the host resource service a similar scenario is applied: available host resources (CPU, memory, and disk space) are periodically measured and their throughputs are benchmarked on harbour start-up.

```
module vagabond2 {
 module services {
  module management {
   module resources {
    module network {
     enum DistanceMetricType {BANDWIDTH, LATENCY, DELAY, JITTER, LOSS, RTT};
     enum StatType {MEAN, MEDIAN, MAX};
     enum CapacityType {TOTAL, AVAILABLE, USED};
     struct Period {Date start; Date end;};

     interface NetworkInfo {
      double getRouteCapacity(in Period p, in StatType st, in CapacityType ct,
                              in DistanceMetricType dmt, in string host1, in string host2)
            raises (vagabond2::NoSuchHostException);
      // ...
     };

     interface NetworkResourceService {
      NetworkInfo getNetworkInfo();
      NetworkResourceAllocator getNetworkResourceAllocator();
     };
    }; // network
   }; // resources
  }; // management
 }; // services
};  // vagbabond2
```

Fig. 7. Vagabond2's Network Resource Service Specification

There is an important difference between finding optimally placed nodes for ADMS data collectors and Web server replicas as described in [42], namely the asymmetric character of the recommendation problem. In ADMS, the connections between data managers and data collectors are different from those between collectors and clients. In the former, data transmission must be error-free and is fairly irresistible against jitter, therefore TCP connections can be used. In the latter, videos must be streamed with limited jitter, but not necessarily error-free (e.g. B frames might be lost), therefore RTP/UDP based communication can be used. A simulation experiment has shown that when many data collectors are near to the client (i.e. low UDP and large TCP communication), the standard deviation (the jitter) remains low, but the total amount of data travelling in the network increases (see Figure 8).

3.2.3. Making an ADMS Component 'Movable'

An adaptive component must be derived from the CORBA interface *AdaptiveApplication* [40, 12]. The key functionality of this interface is the *getApplicationInfo()* method, which is used by a harbour to request for the binaries of the component, and optionally for a set of files it may depend on. Figure 9 illustrates this as an excerpt of Vagabond2's IDL specification.

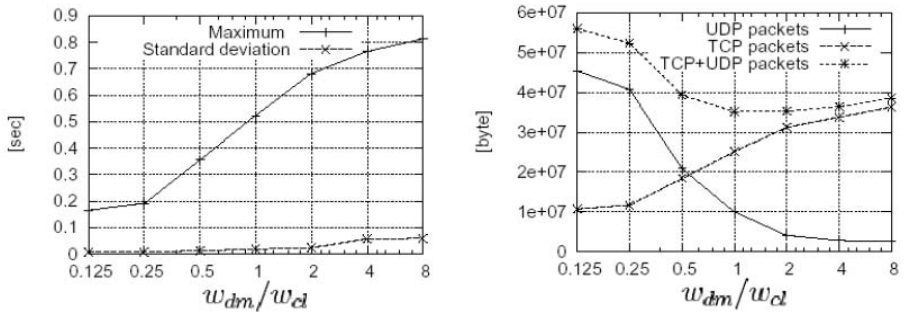


Fig. 8. Jitter (left) and data flow size (right). w_{dm}/w_{cl} is the ratio between the costs of the DC-DM and DC-client links.

```
module vagabond2 {
    // exception and struct declarations ...
    interface ApplicationInfo {
        string getApplicationName();
        string getMainClassName();
        OctetSeq getApplicationJAR() raises (ServerIOException);
        OctetSeq getDependentFilesZIP() raises (ServerIOException);
    };

    interface AdaptiveApplication {
        void start(in StringSeq params) raises (ServiceAlreadyStartedException);
        void stop() raises (ServiceNotStartedException);
        ApplicationInfo getApplicationInfo();
        boolean isIdle();
        ClientSeq getClients();
        void setLocked(in boolean locked);
        boolean isLocked();
    };
    // ...
};
```

Fig. 9. Vagabond2's Core Interfaces for Adaptive Components

Figure 10 illustrates how a data manager component is derived from the *AdaptiveApplication* interface. First, a common abstraction layer is introduced, which is valid for all four ADMS component types. It introduces the interfaces *ADMSServerApplication* and *Session*, allowing the establishment of rate-controlled and transaction-based sessions of certain types (retrieval, acquisition, or management). Second, the bottom layer defines the interfaces and structures comprising the data manager component. An *ADMSDataManager* is used to create so-called data manager sessions (*DMSession*). Each session is associated with exactly one elementary stream. A data manager session provides means to store stripe units for a certain stream segment, to compose a segment tree of known segments, or to retrieve stripe units of a certain segment via a stripe unit iterator. Based on the session's admitted data rate (in kbit/sec), the unit iterator allows to retrieve an according number of stripe units per second.

If an ADMS component is implemented in another language than Java, a Java wrapper component must be added. The binary code has to be carried in the dependent files archive as a shared library. Since an ADMS environment may consist of heterogeneous nodes, the host service provides information about the operating system on a certain harbour, in order to move suitable code to it.

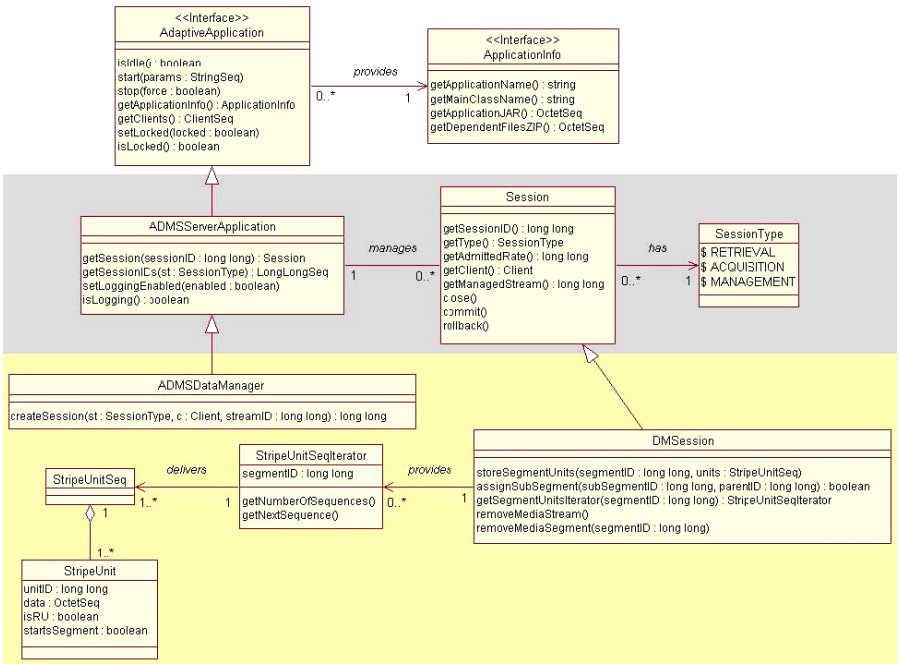


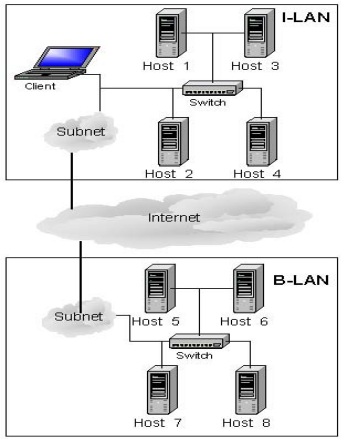
Fig. 10. Data Manager Component as Special Adaptive Application of Vagabond2

3.2.4. Discussion of the Adaptive Architecture

We have evaluated a prototype implementation of our component-based offensive server in the ADMS test bed illustrated in Figure 11. We measured the effect of replicating data managers between two LANs connected via the Internet. One of the LANs was located in Budapest (B-LAN), the other one (I-LAN) in Klagenfurt. The geographical distance of about 500 km assured a ”real” Internet setting. The retrieval client was located in the I-LAN and ran a data collector instance on its own host. Each performance test consists of five test runs and is repeated 50 times. In each run, a sample media stream of 12.5MB size is retrieved from the data manager instances. In test run 0, all data manager instances are running in the remote B-LAN. In test run 1, the data manager from host 8 is replicated to host 4, meaning that one data manager moves ”closer” to the data collector. Both the component code and the requested media stream are replicated, since hosts 1 – 4 are initially

”empty”. In each further test run, one additional data manager is replicated from a host of the remote B-LAN to a host in the local I-LAN.

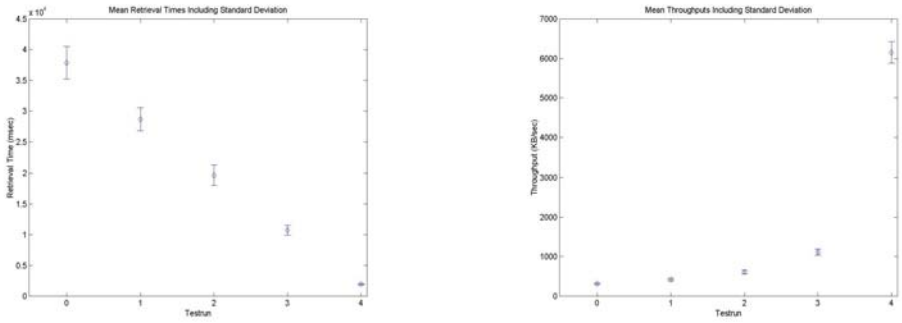
Figure 12 illustrates the mean retrieval times (time needed by the DC to collect the entire media stream) and the mean throughput results (aggregated throughput over all connections from the DC to the DMs). It clearly shows that the more data managers are replicated to the I-LAN, the smaller become the retrieval times, and the higher becomes the throughput. The variation of the retrieval times are quite high as long as most data managers are placed in the remote B-LAN ((a), runs 0–2), they get much smaller as most data managers arrive at the local I-LAN ((a), runs 3–4).



Testrun	DM Hosts
0	Host 5, Host 6, Host 7, Host 8
1	Host 5, Host 6, Host 7, Host 4
2	Host 5, Host 6, Host 3, Host 4
3	Host 5, Host 2, Host 3, Host 4
4	Host 1, Host 2, Host 3, Host 4

Parameter	Value
Stream Size	12,5 MB
Repeated Runs	50

Fig. 11. ADMS Test Bed and Test Scenario

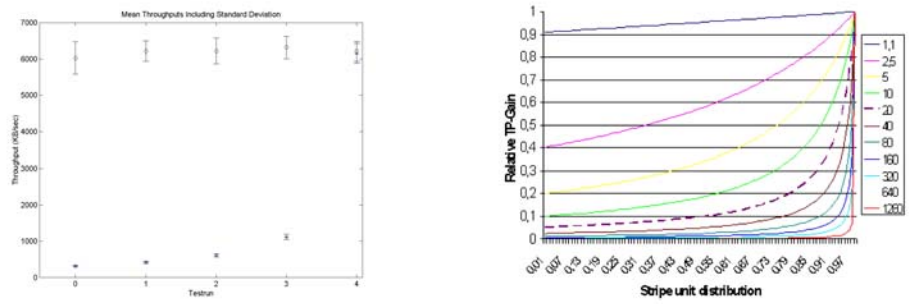


(a) Mean Stream Retrieval Time (b) Mean Throughput

Fig. 12. Performance of Test Runs in the Test Scenario

In Figure 13(a) a head-to-head comparison of throughput is given between replication of four data managers from the B-LAN to the I-LAN (lower measurement series), and replicating them inside the I-LAN (upper series). Not surprisingly, a replication inside a LAN does not make much difference. An interesting property of data managers can be derived from Figure 13(b). It tells

the relative gain on throughput if a certain amount of stripe units is replicated from one LAN to another (including the time for component code replication). If the host recommender of Vagabond2's adaptation service wants to reach 50% overall throughput gain, it has to replicate 95% of the stripe units (actually all data managers) to the LAN of the target data collector. (The different curves show different throughput ratios between the two LANs; more specifically, if T_1 is the throughput when every DM is on the local LAN of the DC, and T_2 is the throughput when every DM is on the remote LAN, the ratio $\frac{T_1}{T_2}$ parameterizes the curves.) The dashed line corresponds to the lower throughput series in Figure 13(a), representing a throughput ratio of 20.



(a) Head-to-head Throughput (b) Relative Throughput Gain

Fig. 13. Measured Throughput in Comparison to Estimated Throughput Gains

Another interesting test run measures the migration times of a data collector component and a minimal adaptive application (Test App) to three harbours. Both components are migrated from their home LAN in Budapest (bme2) (1) to a remote LAN in Klagenfurt (itec), (2) to another LAN at the same campus in Budapest (bme1), and (3) to a node on the same subnet (bme2). The migration of an adaptive component consists of three steps: (1) downloading the byte code and supplementary files, (2) storing them uncompressed on the harbour's file system, and (3) loading the necessary classes while activating the CORBA servant. The harbours have checkpoints at the start and at the end of these steps, and the elapsed times are stored for every started component. Figure 14 shows the results of the experiments. Using such statistical data, the time needed for an adaptive component to migrate can be predicted. From the results we can learn that for a long distance connection the time for download and for activation is nearly equal, but as the distance gets smaller, the activation time gains a higher percentage in the total time (note that the y axes are scaled differently). Storing time is negligible compared to other times (thus, we may ignore the anomaly that storing the small application took more time than storing the bigger one – obviously a consequence of third party load on the bme2 node). The total times show that – not surprisingly – offensive

adaptation over the Internet costs much more than inside a LAN, however it pays much better as well, as shown in Figure 12.

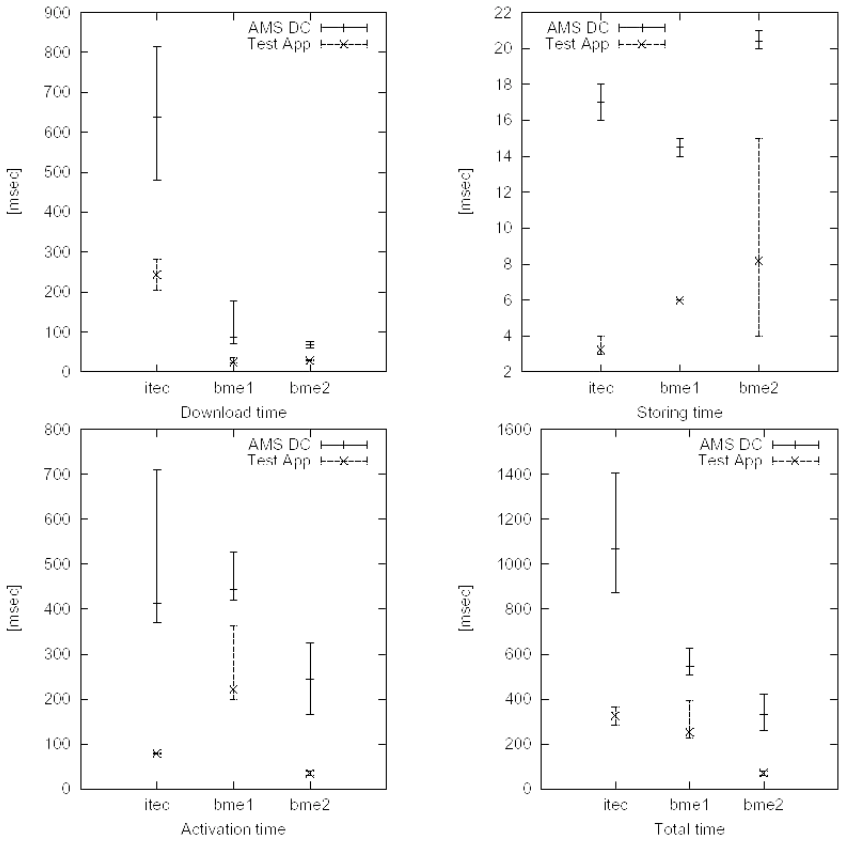


Fig. 14. Elapsed Times for Component Migration

4. Capability-driven Stream-level Adaptation

Whereas ADMS performs offensive adaptation on the number and location of server components, QBIX [1] is an adaptive meta-data hinted proxy that is able to transcode MPEG-4 videos in real-time. This kind of defensive adaptation is applied when e.g. the terminal capabilities of the client indicate that it cannot cope with the quality of the media stream. Media adaptation can be classified into three major categories: bit rate conversion or scaling, resolution conversion, and syntax conversion. Bit rate scaling can adapt to shortages in available bandwidth. Resolution conversion can adapt to bandwidth limitations, but it can also accommodate for known limitations in the

user device, like processing power, memory, or display constraints. Syntax conversion is used in a hybrid network to match sender and client compression protocols.

In contrast to ADMS, QBIX was designed as a single node system. It relies extensively on multimedia standards like MPEG-4 for video coding, MPEG-7 and MPEG-21 for meta-data, and transport or communication protocols like RTP, RTSP and SDP. While older video coding standards did not provide sufficient support for adaptation, MPEG-4 is actually the first standard that offers extensive adaptation options.

4.1. Adaptation in MPEG-4

4.1.1. System-level Adaptation

An MPEG-4 system stream can contain multiple video objects. These video objects may be transmitted with different priorities. Adapting on this level means dropping video objects during transmission (object-based scalability). Besides object-based adaptation, MPEG-4 systems provides spatial, temporal, and SNR fine granular scalability (FGS) support. Combinations of spatial, temporal, FGS, and object-based scalability are possible, although not each combination is allowed (e.g. spatial and FGS). The advantage of adaptation at the system level is that the burden of generating all necessary information for adaptation is in the video production stage. The disadvantage is, however, that possible adaptation options are fixed during encoding and that decoding multi-layer bit streams adds complexity to the decoder.

4.1.2. Elementary Stream Adaptation

Elementary stream (ES) adaptation can be applied on compressed or uncompressed video data. In both cases, adaptation is limited to quality reduction. Adaptation of elementary streams allows for adaptation options not known during the creation of the video in the production stage.

Adaptation on compressed data includes mechanisms for temporal adaptation (frame dropping) and bitrate adaptation (color reduction, low pass filtering, requantization [2–4]). These mechanisms target bit rate adaptation, and can be combined [5].

Finally, adaptation in the pixel domain (on uncompressed video data) is the conventional method for video adaptation. Again, only quality reduction is achievable. Since the video stream is decompressed into raw pixels, which will be encoded again, video adaptation in the pixel domain involves high processing complexity and memory requirements. The advantage of these techniques is flexibility. Video characteristics such as spatial size, color, and bitrate can be

modified. Thus, adaptation in the pixel domain can prepare the video according to client properties for optimal resource usage.

While MPEG-4 offers support for adaptation, the question remains how the proxy determines which adaptation step will give the most benefit in a certain situation. This can only be solved by adding meta-information to a video, as defined by MPEG-7 [9,43].

5. Proxy Architecture

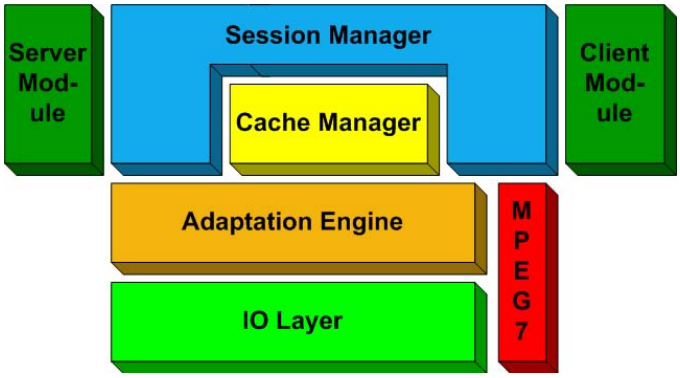


Fig. 15. Proxy Modules

The proxy cache consists of five large modules (Figure 15). The IO Layer is used to read and write video data, the *Adaptation Engine* uses the IO Layer to read/write frames and transforms them. The MPEG-7 module offers means to parse and generate MPEG-7 descriptions. The *Cache Manager* manages the cached videos and uses the adaptation engine to realize its cache replacement strategies. The *Session Management* module consists of three modules: The *Server Module* imitates a media server for the client, the *Client Module* imitates a client for the media server, and the third is the *Session Manager* that controls the video flow.

5.1. IO Layer

The IO layer realizes input/output in the proxy, hiding network and file access behind one abstract class. IO is frame and ES based, i.e., complete frames of an ES are written or read. Currently, raw ES and .mp4 files are supported. On the network, we support multicast and unicast streams packetized with RFC3016 [44]. Advanced packetization layers like MultiSL or FlexMux are features currently not supported; they will be added later. For a detailed description of the mentioned packetization layers, see [45].

5.2. Adaptation Engine

The adaptation engine uses two important concepts:

• Adaptors

An adaptor expects as input a single frame and returns a list of adapted frames. An adaptor is allowed to buffer frames, until it has enough data available to perform one adaptation step. Currently, only visual media adaptors are supported; system level adaptation is not yet, nor is audio. We support most of the adaptations mentioned in Section 4.1.2. The following ones have been implemented:

- Temporal reduction: drop B-frames, or B- and P-frames
- Color reduction
- Spatial reduction
- Bitrate scaling

• Data Channels

A DataChannel reads from an IO object and invokes one of the aforementioned adaptors to (possibly) modify the frame. Due to complex adaptors that might require frame buffering, a list of result frames can be returned. To cope with such a bursty behavior, the DataChannel maintains a send queue where the result frames are inserted. After the adaptation, only one frame per operation is sent to the output IO objects. We allow more than one IO object, so that clients can be grouped together. In the simplest case we have two clients: one viewer and the proxy itself storing the adaptation result to its hard disk.

In the worst case, a buffering adaptor will increase the startup delay, but there should be no additional time penalty afterwards (assuming the proxy is fast enough for real-time adaptation).

The TemporalReduction adaptor is implemented as a compressed domain transcoder. It parses the incoming frames, and according to their frame type it decides to drop complete frames or not. To avoid artifacts in the displayed video, frame dropping follows the following rule: first, drop all B frames within a GOP; if this is not sufficient, drop P frames. I frames are not dropped.

The three remaining adaptors are implemented as pixel domain transcoders, i.e. they decode, change and then encode an MPEG-4 visual ES. The open-source MPEG-4 codec developed in the XviD project (http://www.xvid.org/) is used for transcoding. The encoder behavior is set to constant bit rate (CBR) mode and produces I and P frames only. This reduces the computational complexity significantly, and real-time behavior of the encoder is achieved [1].

To allow for maximum flexibility, all adaptors can be arranged in an *AdaptorChain*. Figure 16 shows an example.

5.3. MPEG-7 Module

The MPEG-7 module adds support for creating and parsing MPEG-7 descriptions. In the current implementation, the focus is on variation descriptors.

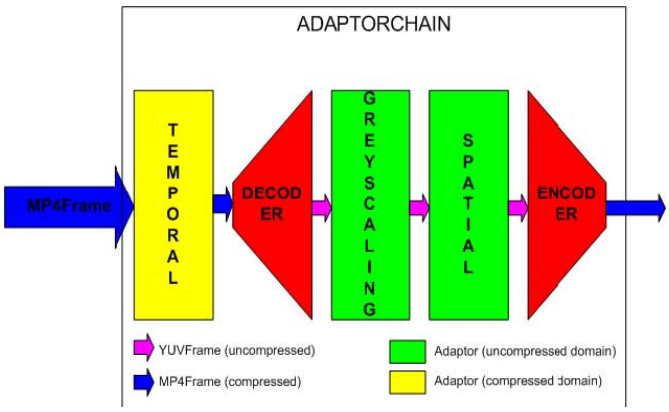


Fig. 16. Example of an AdaptorChain

MPEG-7 describes the internal structure of the source MPEG-4 file including size, type – such as video, audio or BIFS (Binary Interchange Format for Scenes) – and bit rate for each elementary stream. A variation descriptor contains the name of the adaptation step, the expected quality loss and the priority of this adaptation step. Additionally, a modified MPEG-7 description for each elementary stream is generated.

5.4. Cache Management

The Cache Manager (CM) manages all the videos stored in the cache. It uses the adaptation engine to perform the adaptation, and the MPEG-7 module to decide on the adaptation. If no description is available, a default sequence of variations is used. The cache manager is described in detail in [1].

5.5. Session Management

The SessionManagement module imitates a server for the client (*ServerModule*) and a client for the media server (*ClientModule*). It is responsible for creating and managing sessions. A *Session* always connects two communication partners. Thus, two different session types exist:

- Server session: connects client and proxy
- Client session: connects proxy and server

A Session object manages DataChannels. For each ElementaryStream, one DataChannel is created and controlled. A ServerSession manages all data flows from the local proxy disk, a ClientSession all data flows that read their input from the network. An example is given in Section 6.

6. Hybrid Adaptation: A Combined Approach

The integration of the offensive server-level adaptation and the defensive stream-level adaptation is subject of current and future research activities. We have done first steps in integrating the QBIX proxy as a data collector component into the ADMS system. Two major extensions to the QBIX proxy are necessary. First, the proxy has to be made CORBA-aware by being derived from Vagabond2's *AdaptiveApplication* interface. Thus, the proxy must implement the *ADMSDataCollector* interface, as illustrated in Figure 17. As shown for the data manager (Figure 10), the data collector is also derived from the *ADMSServerApplication* interface.

```
#include "common.idl"

module adms {
  module dc {
    interface ADMSDataCollector : ADMSServerApplication {
      boolean announceSession(in long long rtspSessionID, in string clientIP, in string presentationID,
                              in long long streamID, in long long segmentID, in StringSeq dmHosts)
          raises (vagabond2::ServerIOException,
                  adms::common::NoSuchStreamException,
                  adms::common::NoSuchSegmentException,
                  vagabond2::ApplicationLockedException,
                  vagabond2::NoSuchHostException,
                  adms::common::InvalidDataManagerException,
                  adms::dc::NoDataManagerException);
    };
  };
};
```

Fig. 17. Partial IDL Specification of an ADMS Data Collector

The implementation of the *ADMSDataCollector* interface must be a Java-wrapper class, which translates the CORBA method calls into QBIX native calls to control the proxy. The interface implementation must be done in Java since Vagabond2 requires this for instantiating it as an adaptive component in a harbour.

Second, the QBIX IO layer must be enabled to retrieve stripe units from data manager components. This is because an ADMS data collector sees an elementary stream as a sequence of stripe units. In contrast, QBIX - being developed as a single node system - is completely unaware of striping and sees a video as a sequence of frames. Thus, QBIX must use CORBA to

establish data retrieval sessions with the target data managers, and to collect media stripe units.

A *DataConverter* class maps the stripe unit view to a frame view. For the proxy the DataConverter is a black box that implements the IO interface. The two essential methods are *Frame*read(),* and *write(Frame*frm).* Implementing the read method connects QBIX directly with the data manager components. Implementing the write method realizes an ADMS compatible data distributor.

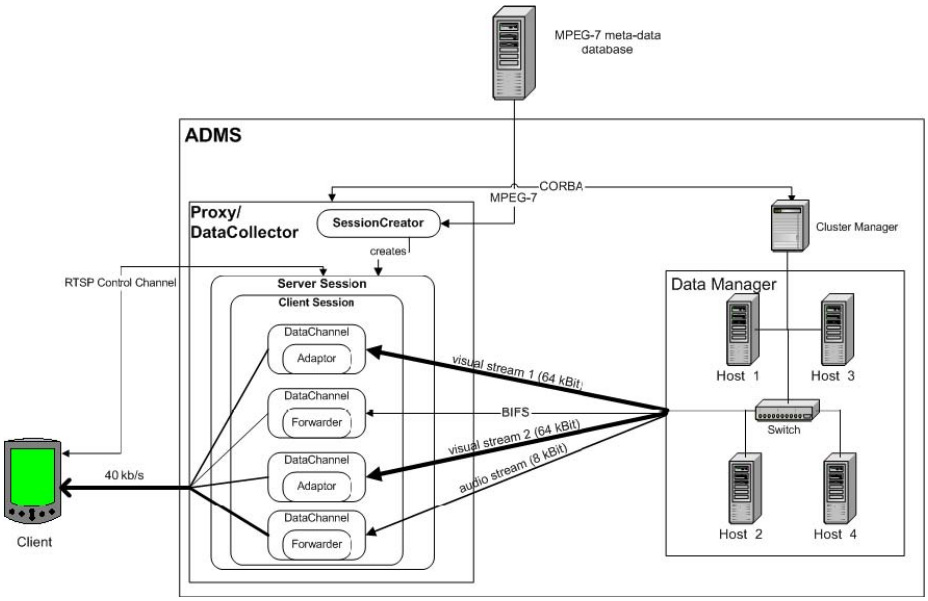


Fig. 18. Example of a Session

An exemplary session for an ADMS server combined with a proxy acting as a gateway is shown in Figure 18. The MPEG-4 video in the example consists of four elementary streams: two visual, one audio, and the BIFS stream (the object descriptor stream is omitted). Assume that only a bandwidth of 40 kbit/sec is available for the client. The source video has an original bit rate of 136 kbit/sec. The media server does not offer a lower bandwidth version. Normally, this would exclude the client from viewing that video.

When initiating a Session, the client sends an RTSP DESCRIBE to the cluster manager (CM). If the cluster manager admits the client, an RTSP REDIRECT is sent back to the client, pointing it to the server node which the cluster manager has chosen as its proxy node. The client resends the DESCRIBE to the proxy node. The DESCRIBE command also contains the client's terminal capabilities, including display size and the 40 kbit/sec bandwidth limitation. In the mean time, the CM has informed the node which part of a video the client has requested and in which quality. If client and CM description do not match, the session is terminated and the CM is informed.

The *SessionCreator* fetches the MPEG-7 data from the meta-database. It then invokes the MPEG-7 module which parses the description and returns a sequence of adaptation steps for each elementary stream of the video. Having seen the video meta-data and the user capabilities, the *SessionCreator* detects the mismatch between the bandwidth of the client and the video. This bandwidth gap is closed by applying an adaptor to each video stream. We assume that a *BitrateScalingAdaptor* is used which reduces the bandwidth from 64 kbit/sec down to 16 kbit/sec for each video stream. The audio stream is simply forwarded. The total bandwidth consumption of the adapted streams is reduced to 40 kbit/sec, so the client can watch a lower quality version of the original video. In the extreme case, it could happen that no video is forwarded at all, only audio (e.g. for clients without a screen).

When the proxy receives a SETUP for each elementary stream, a DataChannel object is created with the aforementioned IO class as input, a *BitRateScalingAdaptor* for the visual streams and an RTP output class. As soon as the client sends a PLAY command, the IO input class starts to fetch the stripe units from the ADMS data manager nodes, converts them to a frame-based view and stores the created frames in an internal buffer. The DataChannel gets the frames, invokes the Adaptor to re-encode them and pushes the adaptation results into the RTP class for sending.

7. Conclusion and Future Work

The requirements, use, and implementation of adaptation techniques in distributed multimedia systems have been discussed. Offensive (or server level) adaptation can proactively cope with situations when the resources of the server nodes and/or network links get saturated. This kind of adaptation is realized by an adaptive distributed multimedia streaming server (ADMS), consisting of four types of adaptive components, which can be combined in an arbitrary number of instances, running on an arbitrary number of server nodes. The resulting virtual streaming server runs on top of Vagabond2, the middleware used for component movement and adaptation support.

Defensive (or stream level) adaptation is realized by varying a media stream's spatial, temporal, or qualitative resolution. QBIX – an adaptive, meta-data hinted proxy – supports this kind of adaptation by transcoding MPEG-4 video elementary streams in real-time. Defensive adaptation is applied for quality-aware cache replacement or when the client has restricted terminal capabilities.

Finally, we have shown how offensive and defensive adaptation can work together in the context of ADMS and QBIX. It has been demonstrated how a QBIX proxy becomes an ADMS data collector component, in order to be replicated and migrated on demand. In this case, the proxy becomes server-aware, with the drawback that it becomes dependent on ADMS, but with the benefit of being optimally controlled by the ADMS cluster manager concerning

physical location and number of instances. A complete implementation and quantitative evaluation of the hybrid adaptation approach is work in progress.

References

1. P. Schojer, L. Böszörményi, H. Hellwagner, B. Penz, S. Podlipnig, Architecture of a Quality Based Intelligent Proxy (QBIX) for MPEG-4 Videos, in: World Wide Web Conference, 2003, pp. 394–402.
2. P. Assuncao, M. Ghanbari, A Frequency-domain Video Transcoder for Dynamic Bit Rate Reduction of MPEG-2 Bit Streams, in: IEEE Trans. on Circuits and Systems for Video Technology, Vol. 8, IEEE Press, 1998, pp. 953–967.
3. Z. Lei, N. Georganas, Rate Adaptation Transcoding for Precoded Video Streams, ACM Multimedia 2002, pp. 127–136.
4. H. Sun, W. Kwok, J. Zdepski, Architectures for MPEG Compressed Bitstream Scaling, in: IEEE Trans. on Circuits and Systems for Video Technology, Vol. 6, IEEE Press, 1995, pp. 191–199.
5. C. Kuhmünch, G. Kühne, C. Schremmer, T. Haenselmann, A Video-scaling Algorithm Based on Human Perception for Spatio-temporal Stimuli, in: Proc. SPIE Multimedia Computing and Networking (MMCN), SPIE Press, 2001, pp. 13–24.
6. L. Böszörményi, M. Döller, H. Hellwagner, H. Kosch, M. Libsie, P. Schojer, Comprehensive Treatment of Adaptation in Distributed Multimedia Systems in the ADMITS Project, in: Proceedings of the 10th ACM International Conference on Multimedia, 2002, pp. 429–430.
7. L. Böszörményi, H. Hellwagner, H. Kosch, M. Libsie, S. Podlipnig, Metadata Driven Adaptation in the ADMITS Project, EURASIP Signal Processing: Image Communication Journal, Special Issue on Multimedia Adaptation, No. 8, September 2003, pp. 749–766.
8. Moving Picture Experts Group, ISO/IEC JTC1/SC29/WG11 N4668: Overview of the MPEG-4 Standard, http://www.chiariglione.org/mpeg/standards/mpeg-4/mpeg-4.htm (2002).
9. Moving Picture Experts Group, ISO/IEC JTC1/SC29/WG11 N5525: MPEG-7 Overview, http://www.chiariglione.org/mpeg/standards/mpeg-7/mpeg-7.htm (2003).
10. J. Y. Lee, Parallel Video Servers: A Tutorial, IEEE Multimedia 5 (2) (1998) 20–28.
11. R. Tusch, C. Spielvogel, M. Kröpfl, L. Böszörményi, An Adaptive Distributed Multimedia Streaming Server in Internet Settings, in: SPIE Proceedings of Information Technologies and Communications (ITCom), Internet Multimedia Management Systems IV, September 2003, pp. 312-323.
12. R. Tusch, Towards an Adaptive Distributed Multimedia Streaming Server Architecture Based on Service-oriented Components, in: Joint Modular Languages Conference (JMLC), 2003, pp. 78–87.
13. D. W. Brubeck, L. A. Rowe, Hierarchical Storage Management in a Distributed VOD System, IEEE Multimedia 3 (3) (1996) 37–47.

14. W. Bolosky, J. Barrera, R. Draves, R. Fitzgerald, G. Gibson, M. Jones, S. Levi, N. Myhrvold, R. Rashid, The Tiger Video Fileserver, in: 6th International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV), 1996, pp. 97–104.

15. J. Gafsi, U. Walther, E. W. Biersack, Design and Implementation of a Scalable, Reliable, and Distributed VOD-Server, in: Proceedings of the 5th joint IFIPTC6 and ICCC Conference on Computer Communications, 1998.

16. Helix Community, The Helix Platform, URL: https://www.helixcommunity.org/2002/intro/platform (2002).

17. RealNetworks, Inc., Helix Universal Server Administration Guide, URL: http://docs.real.com/docs/HelixServer9.pdf (2003).

18. J. Dilley, B. Maggs, J. Parikh, H. Prokop, R. Sitaraman, B. Weihl, Globally Distributed Content Delivery, IEEE Internet Computing 6 (5) (2002) 50–58.

19. S. Saroiu, K. P. Gummadi, R. J. Dunn, S. D. Gribble, H. M. Levy, An Analysis of Internet Content Delivery Systems, in: 5th Symposium on Operating Systems Design and Implementation, 2002, pp. 315–327.

20. S. Saroiu, K. P. Gummadi, S. D. Gribble, Measurement Study of Peer-to-Peer File Sharing Systems, in: SPIE/ACM Proceedings of Multimedia Computing and Networking, Vol. 4673, 2002, pp. 156–170.

21. D. Xu, M. Hefeeda, S. Hambrusch, B. Bhargava, On Peer-to-Peer Media Streaming, in: 22nd International Conference on Distributed Computing Systems (ICDCS), 2002, pp. 363–371.

22. D. A. Tran, A Peer-to-Peer Architecture for Media Streaming, IEEE JSAC Special Issue on Advances in Overlay Networks 21 (10), to appear.

23. J. Zinky, D. Bakken, R. Schantz, Architecture Support for Quality of Service for CORBA Objects, Theory and Practice of Object Systems 3 (1).

24. D. C. Schmidt, D. L. Levine, S. Mungee, The Design of the TAO Real-Time Object Request Broker, Computer Communications, Elsevier Science 21 (4) (1998) 294–324.

25. A. Hafid, G. Bochmann, An Approach to QoS Management in Distributed Multimedia Applications: Design and Implementation, Multimedia Tools and Applications 9 (2) (1999) 167–191.

26. P. Francis, S. Jamin, V. Paxson, L. Zhang, D. F. Gryniewicz, Y. Jin, An Architecture for a Global Internet Host Distance Estimation Service, in: IEEE INFOCOM, 1999, pp. 210–217.

27. W. Theilmann, K. Rothermel, Dynamic Distance Maps of the Internet, in: IEEE INFOCOM, 2000.

28. D. Andersen, H. Balakrishnan, M. Kaashoek, R. Morris, Resilient Overlay Networks, in: Proceedings of the 18th ACM Symposium on Operating Systems Principles, 2001, pp. 131–145.

29. T. S. E. Ng, H. Zhang, Predicting Internet Network Distance with Coordinates-based Approaches, in: IEEE INFOCOM, 2002, pp. 170–179.

30. J. Waldo, The Jini Architecture for Network-centric Computing, Communications of the ACM 42 (7) (1999) 76–82.

31. R. Friedman, E. Biham, A. Itzkovitz, A. Schuster, Symphony: An Infrastructure for Managing Virtual Servers, Cluster Computing 4 (3) (2001) 221–233.

32. J. Kangasharju, F. Hartanto, M. Reisslein, K. W. Ross, Distributing Layered Encoded Video through Caches, in: Proceedings of IEEE INFOCOM, 2001, pp. 622–636.

33. S. Paknikar, M. Kankanhalli, K. R. Ramakrishnan, S. H. Srinivasan, L. H. Ngoh, A Caching and Streaming Framework for Multimedia, in: Proceedings of ACM Multimedia, 2000, pp. 13–20.

34. M. Sasabe, N. Wakamiya, M. Murata, H. Miyahara, Proxy Caching Mechanisms With Video Quality Adjustment, in: Proceedings of the SPIE Conference on Internet Multimedia Management Systems, 2001, pp. 276–284.

35. R. Rejaie, J. Kangasharju, Mocha: A Quality Adaptive Multimedia Proxy Cache for Internet Streaming, in: 11th International Workshop on Network and Operating Systems Support for Digital Audio and Video, 2001, pp. 3–10.

36. S. Podlipnig, L. Böszörményi, Replacement Strategies for Quality Based Video Caching, in: IEEE ICME 2002 International Conference on Multimedia and Expo, 2002, pp. 49–52.

37. Moving Picture Experts Group, ISO/IEC JTC1/SC29/WG11 N5231: MPEG-21 Overview, http://www.chiariglione.org/mpeg/standards/mpeg-21/mpeg-21.htm (2002).

38. Internet Engineering Task Force, RFC 2326: Real Time Streaming Protocol (RTSP), http://www.ietf.org/rfc/rfc2326.txt (1998).

39. Object Management Group, Common Object Request Broker Architecture: Core Specification, 3rd Edition, http://www.omg.org/cgibin/doc?formal/02-12-02.pdf (2002).

40. B. Goldschmidt, R. Tusch, L. Böszörményi, A Mobile Agent-based Infrastructure for an Adaptive Multimedia Server, in: 4th Austrian-Hungarian Workshop on Distributed and Parallel Systems (DAPSYS), 2002, pp. 141–148.

41. B. Goldschmidt, Z. Laszlo, A Proxy Placement Algorithm for the Adaptive Multimedia Server, in: Proceedings of the 9th International Euro-Par Conference, 2003, pp. 1199–1206.

42. L. Qiu, V. N. Padmanabhan, G. M. Voelker, On the Placement of Web Server Replicas, in: IEEE INFOCOM, 2001, pp. 1587–1596.

43. ISO/IEC, FDIS 15938-5 – MPEG-7 Standard - Multimedia Description Schemes (2001) 539.

44. Internet Engineering Task Force, RFC 3016: RTP Payload Format for MPEG-4 Audio/Visual Streams, http://www.ietf.org/rfc/rfc3016.txt (2000).

45. M. Ohlenroth, H. Hellwagner, RTP-Packetization of MPEG-4 Elementary Streams, in: IEEE ICME 2002 International Conference on Multimedia and Expo, 2002, pp. 465–468.

Roland Tusch received the M.Sc. degree in computer science in 1999 from the University Klagenfurt, Austria. Since 2000, he has been working as an University Assistant with the Institute of Information Technology at the University Klagenfurt. He has also been working on his Ph.D. in the area of adaptive distributed multimedia servers since then. He is designer and co-developer of Vagabond2 (a QoS-aware middleware for adaptive servers) and ADMS (the Adaptive Distributed Multimedia Streaming Server built on top of Vagabond2). His research interests include distributed multimedia systems, QoS-aware middleware systems, and standards for multimedia communication. He is an associate member of IEEE.

László Böszörményi is a full professor and the head of the Department of Information Technology at the University Klagenfurt, Austria. He is a member of ACM, IEEE and OCG, he is deputy head of the Austrian delegation at the Moving Picture Experts Group (MPEG, the ISO/IEC JTC1/SC29 WG11).

In his research, he is focusing currently on Adaptation in Distributed Multimedia Systems. He is leading a number of projects in this area, such as the QBIX (Adaptive, *Q*uality-*B*ased *I*ntelligent Video Pro*x*y used as a gateway and as a cache), the ADMS (*A*daptive *D*istributed *M*ultimedia *S*erver, based on a mobile agent based infrastructure) and the Calm-Video (Video applications with a large number of concurrent videos in different quality) projects. He is author of several books, he publishes regularly in refereed international journals and conference proceedings. He has been organising several international conferences and workshops.

Balazs Goldschmidt received the M.Sc. degree in technical informatics in 1999 from the Budapest University of Technology and Economics, Hungary. Since 1999 he has been a PhD student at the university's Department of Control Engineering and Information Technology, his topic is real-time distributed object-oriented systems. Since 2001 he has been working at the department as an assistant lecturer. His research interest include distributed object-oriented systems and QoS-aware distributed resource brokering. He is a co-developer of Vagabond2.

Hermann Hellwagner is a full professor of computer science at the Department of Information Technology, University Klagenfurt, Austria. He is a member of the IEEE, GI and OCG as well as the head of the Austrian delegation to the Moving Picture Experts Group (MPEG – ISO/IEC JTC1/SC29/WG11).

His current research areas are distributed multimedia systems, multimedia communications, and Internet QoS. Current projects are on digital video communication, a streaming protocol supporting media adaptation, and multimedia bit stream description techniques within the MPEG-21 Digital Item Adaptation (DIA) standardization effort. He is the editor of several books, has published widely on parallel computer architecture and parallel programming, and now publishes regularly on multimedia communications and adaptation in refereed journals and conference proceedings. He has organized several international conferences and workshops.

Peter Schojer received the M.Sc. degree in computer science in 2001 from the University Klagenfurt, Austria. Since 2001, he has been working on his Ph. D. in the area of adaptive video caching. His main research interests are proxy caching, video transcoding and the use of meta-data in the context of proxy caching and video streaming. He is designer and one of the main developers of ViTooKi (the library used by QBIX), an open-source implementation of a multimedia framework supporting adaptive, error-resilient streaming of MPEG-4 videos. The source code for ViTooKi is available at http://vitooki.sourceforge.net.