Towards an Adaptive Distributed Multimedia Streaming Server Architecture Based on Service-oriented Components

Roland Tusch

Institute of Information Technology, Klagenfurt University, Austria
roland@itec.uni-klu.ac.at

Abstract. This paper presents an adaptive distributed multimedia streaming server architecture (ADMS) which explicitly controls the server-layout. It consists of four types of streaming server components, which all provide dedicated services in an arbitrary number of instances on an arbitrary number of server hosts. Vagabond2 is used as the underlying middleware for component adaptation. It is shown, how the CORBA-based components have to be declared in order to run on top of Vagabond2. Finally, inter-component dependencies are pointed out, which have to be taken into account during component adaptations.

1 Introduction

Current distributed multimedia servers are monolithic and performance-optimized in order to cope with thousands of simultaneous client requests. However, in case of increasing numbers of clients in heterogenous environments, it is usually not the server, but the network that becomes a bottleneck. Since server implementations are not component-based, they have no chance to cope with this problem and might reject client requests due to network resource shortages. From the client's point of view, one solution to this problem is the usage of a proxy-server. However, this approach has only limited power, as the server has no explicit control over client-side proxies.

Component-oriented programming has shown to be a major improvement in implementing complex distributed systems, allowing to build independent pieces of software that can be reused and combined in different ways. A number of software components for building distributed server applications like distributed web services based on Enterprise JavaBeans[1], DCOM[2], the .NET framework, or CORBA's Component Model (CCM)[3], exists. Much less effort has been done in building components for distributed multimedia streaming services. There are two key differences between multimedia and non-multimedia services. First, a multimedia streaming service imposes real-time constraints on delivering media

data to the clients. Second, the amount of generated network traffic and the time periods for serving clients are usually orders of magnitude bigger than in the case of non-multimedia services. This is why existing components originally built for distributed web services usually can not be reused for building distributed multimedia services.

Components for real-time server applications also require a middleware that enables for guaranteed or predicted component adaptation times, i.e. times for migrating or replicating a server component from one server host to another. Currently, there is no such middleware system available for operation in internet settings. Existing systems like e.g. Jini[4] or Symphony[5] only provide means for best-effort component adaptations. They rely on the existence of a network of reasonable speed and latency. The adaptive distributed multimedia streaming server architecture proposed in [6], which serves as the fundament for this work, is based on Vagabond2[7]. Vagabond2 is a CORBA-based middleware that enables for dynamic adaptations of so-called *adaptive applications*. An adaptive application is a service-based component which provides the necessary information to be run and managed on a remote Vagabond2 host. Similar to the Network QoS Broker of the MASA system in mobile environments[8], Vagabond2 bases on a resource broker component which translates client request demands into network and host resource demands. Network and host resource availabilities of all Vagabond2 server hosts are measured periodically and the measured data is used for statistical predictions of resource abilities in cases of required component adaptations.

This paper explores the building blocks of an adaptive distributed multimedia streaming server architecture, called ADMS[6]. It presents the minimum decomposition of a distributed streaming server architecture, which yet allows to construct flexible demand-oriented streaming services in internet settings. Each of the four distinguished CORBA-based components provides clearly defined interfaces and can be combined in an arbitrary number, allocated on a dynamic number of server nodes. The number of dependent component instances defines a virtual multimedia streaming server cluster. Its size may grow and shrink dynamically over time, depending on QoS requirements derived from client requests, as well as host and network resource availabilities.

2 Service-oriented Components of a Distributed Multimedia Streaming Server

Before discussing the server components of an adaptive distributed multimedia streaming server in detail, a brief connection to the ADMS runtime environment including its service-based middleware Vagabond2 is provided.

2.1 AdaptiveApplication: The Service-oriented Component Interface of Vagabond2

The runtime environment for an adaptive multimedia streaming server could look like the one illustrated in figure 1, as proposed in [6]. It provides means for non-real-time component management, and real-time component interactions.

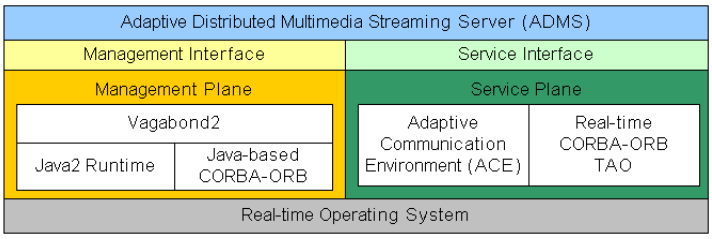

Fig. 1. The ADMS Runtime Environment

The component-based streaming server ADMS builds on top of a management plane and a service plane. The management plane interfaces to Vagabond2, which itself requires its runtime environment including a Java-based CORBA ORB for performing best-effort component adaptations. On the other hand, the service plane allows server components to directly interact by taking into account real-time constraints. Thus, the service plane is constituted by the native component implementations based on e.g. ACE and the TAO real-time CORBA ORB[9].

Vagabond2 allows to specify an adaptive service component by deriving it from the CORBA interface `AdaptiveApplication`[7, 6]. Here the terms application and service are considered synonymously. The key functionality of this interface is the `getApplicationInfo()` method, which allows to dynamically request the binaries of the service component, as well as a possibly dynamic set of files, upon which the component depends on.

Figure 2 illustrates this connection as an excerpt of Vagabond2's IDL specification.

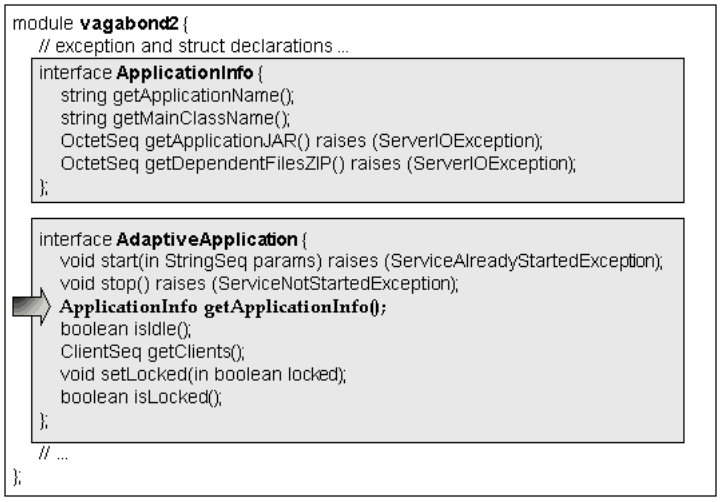

Fig. 2. Vagabond2's Core Interfaces for Adaptive Service Components

The `getApplicationJAR()` method indicates that Vagabond2's implementation is Java-based, allowing for a dynamic replication/migration of the component based on byte code. As a result, each adaptive service component has to provide a Java implementation of its specification. However, using JNI, the service of the component can use a native implementation of it in order to handle real-time issues.

2.2 The Building Blocks of an ADMS

When designing an adaptive distributed streaming server, the following two guidelines have to be taken into account. First, each adaptive component must be *independent* to a certain extent, *reusable*, *adaptable*, *combinable*, and *movable*. And second, a component should fulfill a complete dedicated task for a certain usage scenario. E.g. during a data acquisition scenario a media stream has to be stored on a set of server nodes. To perform this operation in a distributed environment, one component is needed to receive the media stream, split the stream into smaller pieces, and 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 run on separate server nodes. The second guideline implies a beneficial side-effect that the number of distinct components is kept small, since the number of usage scenarios is quite limited.

Following these guidelines, four basic building blocks have been identified to constitute an ADMS: *data distributors* (DDs), *data managers* (DMs), *data collectors* (DCs), and *cluster managers* (CMs). Each of these components can be replicated or migrated on demand, and provides services to other components. Figure 3 demonstrates a sample ADMS consisting of two DDs, four DMs, two DCs, and one CM. In a typical ADMS environment there is only one CM instance, but a number of DD, DM, and DC instances, where each instance should run on a dedicated host.

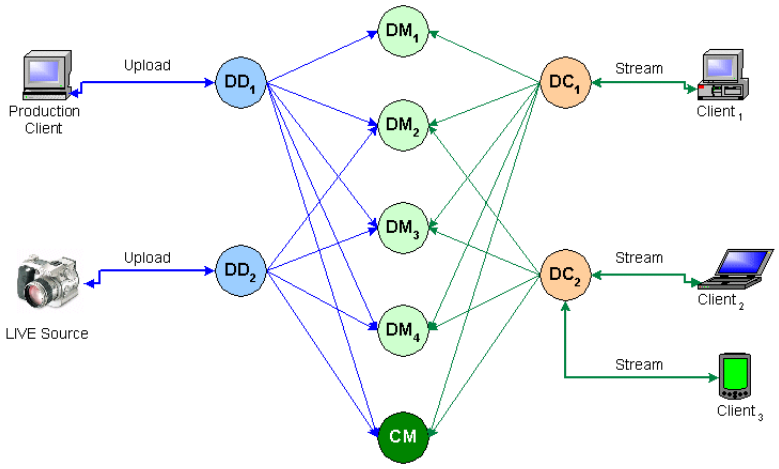

Fig. 3. Service-oriented Components Building an ADMS system

2.2.1 Data Distributor: A data distributor is a service-based component that distributes media data received from a production client or a live camera to a selected couple of data managers. The unit of distribution is a so-called *stripe unit*, which either can be of constant data length (CDL), or constant time length (CTL). Like in RAID level 5 systems parity units are generated, in order to cope with data manager failures. In cases of a non-live source, the process of distribution can be driven by a MPEG-7 document describing a temporal decomposition of the me-

dia stream. Thus, a media stream can be decomposed into a number of segments, organized in an arbitrary number of levels.

Depending on the number of data managers the media stream is striped across, the stream is either single, narrow, or wide striped. The number and location of target data managers is advised by the cluster manager component. 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. In this case, byte boundaries of elementary stream segments have to be adapted in a possibly supplied MPEG-7 descriptor.

2.2.2 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 of them. Since one elementary stream or segment may be striped among a number of data mangers, each data manager only stores a portion of the stream or segment. Figure 4 gives an insight about how a data manger internally organizes its managed media streams. It consists of a set of partial media streams, which themselves consist of a set of leaf and compound media segments. Only leaf media segments store real media data. Stream segmentation is supported to perform more efficient media data buffering.

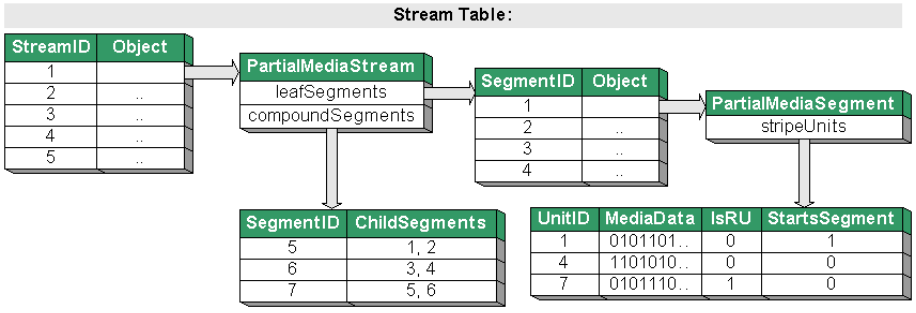

Fig. 4. Objects Comprising a Data Manager Component

Figure 5 illustrates, how a data manager is derived from Vagabond2's `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 to create rate-controlled and transaction-based sessions of certain type (retrieval, ac-

quisition, or management) with the component. Second, the bottom layer defines the interfaces and structures comprising the data manager component. An `ADMSDataManager` allows only to create so-called data manager sessions (`DMSession`). Since each session is associated with exactly one elementary stream, a data manager session provides means to e.g. store stripe units for a certain stream segment, to compose a segment tree of known segments by the component, or to retrieve stripe units of a certain segment via a stripe unit iterator. Based on the sessions admitted data rate (in kbit/sec), the unit iterator allows to retrieve an according number of stripe units per second.

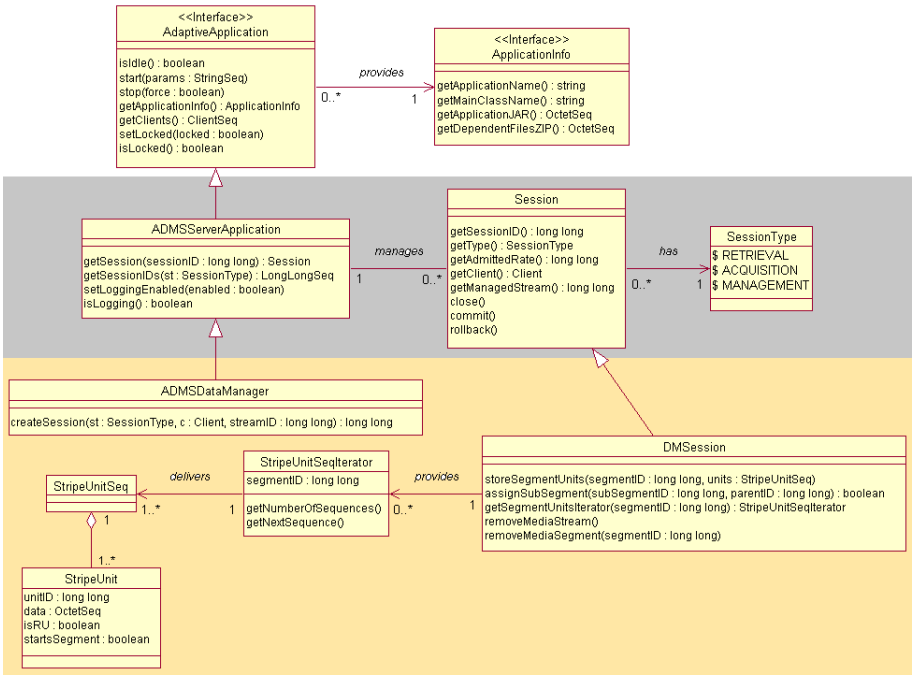

Fig. 5. The Data Manager Component as Special Adaptive Application of Vagabond2

2.2.3 Data Collector: A data collector component performs the inverse operations of a data distributor. Its main task is to serve a client request by collecting stripe units of a certain media stream from a given set of data managers, resequencing those units, and sending the buffered stream to the client using RTP. It provides server-level fault tolerance by exploiting parity units in cases of unavailable data managers.

Since collecting stripe units from the associated data managers is always a load-generating and time-consuming task, the data collector should also include a caching component, in order to reduce startup latencies. Following Lee's framework of video distribution architectures[10], the data collector can implement the *independent-proxy* or *proxy-at-client*-model. However, there are advantages of running the data collector component as a service on a separate Vagabond2 host. First, the client does not know anything about the internal ADMS layout. Second, the collector can serve a set of clients with even unequal client capabilities by applying gateway functionalities, e.g. by transcoding the streams. This is illustrated in figure 3 for DC_2, which streams the same media stream with different qualities to a notebook and to a handheld client simultaneously. And finally, keeping the data collector component independent from the clients allows the cluster manager to select the most appropriate one for serving a certain client request.

2.2.4 Cluster Manager: Since a typical ADMS environment consists of a dynamic number of DDs, DMs, and DCs, one central component is required for managing these component instances and compositing the virtual server. The cluster manager component does this and furthermore implements a directory service keeping track which data mangers store data of which elementary streams, and which data collectors keep which elementary streams in their cache. For data acquisition it advises data managers for stream distribution by applying load-balancing strategies for network and host resources. For data retrieval it originally handles client requests based on RTSP, and delegates the request to appropriate data collectors, if possible. These decisions are based on server loads, network loads, client vicinity to data collectors, and client terminal capabilities.

3 Dependence Management Between ADMS Components

Similar to the dependence management of component-based applications in operating systems suggested in [11], a dynamic dependence management between ADMS components is necessary. Despite the obvious dependency of distributors and collectors from the cluster manager, a collector may be faced with dynamic, stream-based dependencies from data managers. In particular, a data collector dc depends on a data manager dm regarding a media stream m, iff dc does not have cached m, and dm stores at least a portion of m (m_p). Speaking in terms of a component configurator as presented in [11], the hook from dc to dm is dynamic,

since it is removed, if either dc caches m, or dm is replaced by another data manager dm_r. The replacement itself can either be accomplished by replicating the partial media stream m_p to an existing data manager, or to a newly created data manager, or even by replicating the whole data manager dm to a new host.

Different dependence configurations may result in considerably different performance behaviors when serving a client request. This is especially the case, if the locations of a data collector's hooks are wide spread. Performance evaluations in the ADMS testbed, consisting of servers in two LANs interconnected by the internet with widely distributed data managers, have shown similar results. For detailed results the interested reader is referred to [7]. It has to be noted, that the dependence management is accomplished by the cluster manager. The CM is also responsible for taking care of optimal dependencies between data collectors and data managers. Thus, if it recognizes increased request denials due to bad collector hooks, it has to reconfigure the server layout by replicating/migrating media streams and/or ADMS server components.

4 Conclusion and Future Work

The major contribution of this paper lies in exploring and designing the building blocks of an adaptive distributed multimedia streaming server. Current distributed multimedia servers have deficits in adapting themselves to changing client demands due to their monolithic implementations. Four basic components have been identified that allow to compose a distributed streaming server, which is able to adapt itself in reaction to varying client demands. Furthermore, it is shown how these components are built on top of Vagabond2, which serves as the middleware for component management and adaptation. The contribution is concluded by pointing out the explicit inter-component dependencies, which have to be considered during server adaptations.

A summarization regarding the common values of most important properties of ADMS components is given in table 1. While a data manger e.g. is highly adaptable due to the coding characteristics and the amount of the media data it stores, a cluster manager is much less adaptable. On the other hand, a cluster manager is much more mobile than a data manager, since it depends on orders of magnitude less data.

Currently a resource broker component is being developed which allows to monitor and manage network and host resources between all server hosts. This component will be used by the cluster manager to decide

Component	Dependencies	Adaptability	Combinability	Mobility
CM	none	low	DD, DC	high
DD	CM, DM	medium	CM, DM	high
DM	none	high	DD, DC	low
DC	CM, DM	medium	CM, DM	medium

Table 1. Values for Most Important Properties of ADMS Components

whether a client request will be admitted or not. Future plans are to integrate a native implementation of a MPEG-4 proxy server as a data collector component in the ADMS system. Since this proxy server comprises gateway functionality, the data collector will even be able to adapt media streams to different qualities. This approach opens a new research area concerning combined adaptation possibilities regarding server adaptations and media stream adaptations.

References

1. Sun Microsystems: Enterprise JavaBeans Specification. 2.1 edn. (2002) http://java.sun.com/products/ejb/.
2. Microsoft: DCOM Technical Overview. (1996) http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dndcom/html/msdn_dcomtec.asp.
3. Object Management Group: CORBA Components. 3.0 edn. (2002) http://www.omg.org/cgi-bin/doc?formal/02-06-65.pdf.
4. Waldo, J.: The Jini Architecture for Network-centric Computing. Communications of the ACM **42** (1999) 76–82
5. Friedman, R., Biham, E., Itzkovitz, A., Schuster, A.: Symphony: An Infrastructure for Managing Virtual Servers. Cluster Computing **4** (2001) 221–233
6. Tusch, R.: AMS: An Adaptive Multimedia Server Architecture. Technical Report TR/ITEC/02/2.06, Institute of Information Technology, Klagenfurt University (2002)
7. Goldschmidt, B., Tusch, R., Böszörményi, L.: A Mobile Agent-based Infrastructure for an Adaptive Multimedia Server. Parallel and Distributed Computing Practices, Special issue on DAPSYS 2002 (2003) To appear. Papers is also available as technical report TR/ITEC/03/2.05.
8. Carlson, D., Hartenstein, H., Schrader, A.: QoS Orchestration for Mobile Multimedia. In: Proceedings of ASW'2001, The First Workshop on Applications and Services in the Wireless Networks. (2001)
9. Schmidt, D.C., Levine, D.L., Mungee, S.: The Design of the TAO Real-Time Object Request Broker. Computer Communications, Elsivier Science **21** (1998)
10. Lee, J.Y.: Parallel Video Servers: A Tutorial. IEEE Multimedia **5** (1998) 20–28
11. Kon, F., Campbell, R.H.: Dependence Management in Component-Based Distributed Systems. IEEE Concurrency **8** (2000) 26–36