A MOBILE AGENT-BASED INFRASTRUCTURE FOR AN ADAPTIVE MULTIMEDIA SERVER

Balázs Goldschmidt
Budapest University of Technology and Economics
Department of Control Engineering and Information Technology
balage@inf.bme.hu

Roland Tusch, László Bőszörményi
Klagenfurt University
Institute of Information Technology
{roland, laszlo} @tec.uni-klu.ac.at

Abstract
This paper introduces a mobile agent-based infrastructure for an adaptive multimedia server enabling a dynamic migration or replication of certain multimedia applications among a set of available server nodes. It discusses the requirements from both, the server’s and the middleware’s point of view to each other and comes up with a specification and implementation of a CORBA-based interface between them.

Introduction
In [11] we presented a distributed multimedia server architecture, which builds upon the concepts for distributed multimedia servers described in [7]. The key extension of the presented architecture therein is the capability that server components (also referred to as applications) may be dynamically migrated or replicated to other server nodes on demand. Program and data dependencies are also considered. The proposed adaptive multimedia server architecture (AMS) mainly consists of three applications: the Cluster-Manager, the Data-Manager and the Data-Collector. The AMS cluster manager controls the layout of a server cluster concerning the locations of the two other types of applications, namely the data managers and data collectors (see figure 1). Data managers are applications providing facilities for storing and retrieving units of media data, the so-called stripe units. And finally, the data collector’s task
is to collect the stripe units from the data nodes, to reassemble and to stream them out to the requesting clients.

Since the proposed architecture employs the full server utility model [10], each server node can run at most one application at a time. Following this model and focusing on especially the data collector application, there are two main questions regarding an optimal provision of quality of service to the requesting clients: (1) Does the application run on optimal locations? (2) If not, what locations should be recommended to run the application and what are the costs for performing an application migration or replication? These questions can typically not be answered by the server nodes themselves, since they require both local and global views on the performance behavior of the distributed server architecture. Thus, a kind of infrastructure is required for monitoring the resource usage on each server node, as well as the quality of the connection links between the nodes.

Infrastructures for managing an adaptive multimedia server architecture are rather spare. One candidate seemed to be Symphony [3], which represents a service-based management infrastructure for executing virtual servers in internet settings. However, Symphony does not support replication/migration of data objects associated with a multimedia service. QoS-aware middleware frameworks supporting QoS-driven adaptation issues, as presented in [8], focus on resource management dynamics, rather than on application-level dynamics. In [1] it is illustrated that mobile agents are very well suited for network management issues like fault, configuration and performance management. Monitoring application demands, network and server loads are predestined tasks for a mobile agent-based middleware. These lead us to the idea of using such middleware for the adaptive multimedia server. Since the process of replication/migration is not to be intended to happen in real-time, an extended version of the CORBA-based mobile agent system Vagabond [4] is used. Vagabond is a 100% pure Java mobile agent system, first developed to be used in distributed multimedia database systems. However, its rather simple design allows to modify and use it as a middleware for the adaptive multimedia server, resulting in its successor Vagabond2.

This paper addresses the requirements of the adaptive multimedia server (AMS) to the underlying infrastructure (Vagabond2), as well as the needs of Vagabond2 to the adaptive server. The remainder of this paper is organized as follows. Section 1 discusses the requirements of the AMS to the underlying infrastructure. In section 2 the needs of Vagabond2 to AMS are disputed. Section 3 presents the CORBA interface specification between AMS and Vagabond2. Section 4 deals with the internal architecture and implementation of Vagabond2, and section 5 concludes this paper including a plan on future work.
1. Needs to the Underlying Infrastructure

As described in [11], the key objectives of an adaptive multimedia server are (1) to maximize the number of acceptable client requests, (2) to minimize startup delays for those clients and (3) thereby to reduce network traffic during the transmission of the requested media streams. These goals can be achieved by applying a number of different adaptation strategies, either separately or combined. In order to perform these adaptation strategies, a set of services has to be provided by the underlying infrastructure, which can be divided into two major classes: application services and adaptation services.

Consider figure 1, where an AMS cluster $C_j$ covers a set of nodes $M$ either being idle or running a certain server application. Each node is equipped with the AMS runtime environment. Consider also to be given 1 node being reserved for the Cluster-Manager application, $k$ data nodes running the Data-Manager application, and $l$ nodes running the Data-Collector application. ($k + l + 1 < M$ must hold). The application service of the underlying infrastructure has to provide means for replicating or migrating a certain server application from a host to one of the $M - k - l - 1$ remaining hosts, which are in an idle state. This is illustrated on figure 1 by replicating the Data-Collector application to cluster node $CN_{M-1}$. Since a multimedia application may be dependent on program and data files, it also has to provide facilities for carrying those dependencies during the replication/migration processes. The infrastructure needs not to support migration or replication of stateful applications in real-time.

Figure 1. Replication of the Data-Collector application to the idle node $CN_{M-1}$

The adaptation services have to provide means for automatically managing an application based on a certain adaptation policy. Furthermore, recommen-
ations for allocating a new server node for a certain application in the case of a required adaptation have to be provided. Thus, the adaptive media server should also be able to delegate the adaptation control over a certain application (typically the Data-Collector application) to the underlying middleware. On the other hand, the given recommendations have to take into account local server information (as e.g. CPU, network, disk and memory usage) of each server node, global information concerning the quality of the network connections between the server nodes within the cluster, as well as the topological distances and QoS parameters of the requesting clients served by the cluster.

2. Needs by the Underlying Infrastructure

The major problem for the middleware - concerning the requirements of the adaptive server - is the one of recommending a set of server nodes for a required application-adaptation. The data collectors described in section 1 are distributed in the network using the services of Vagabond2. One central component of the infrastructure, namely the so-called host recommender, should recommend host machines (Harbours) where these applications should be loaded. In particular, the host recommender should (1) provide recommendations concerning the set of hosts that should be used by the data collectors, (2) give hints on the costs (in time) for a certain adaptation, and finally (3) dynamically adapt an already loaded application according to the previous two offers.

The problem of recommending the optimum set of server nodes can be modelled as a capacitated facility location problem (or $k$-median problem, where $k$ is a given constraint). In [6] it is shown that this problem is NP-hard for general graphs. However, several approximation algorithms have been studied recently and the best published approximation factor to date is 4, with a running time of $O(N^3)$ [2].

In order to perform approximative recommendations for application locations the host recommender needs information about the server nodes, the network, as well as the current and future set of requesting clients. Concerning the server nodes the infrastructure itself can monitor the usage of the server’s resources (CPU-load, disk and memory usage). Regarding the quality of the network links between the server nodes a network monitoring agent can be used to periodically perform observations on network throughput and latency times. The major requirements of Vagabond2 to the managed multimedia applications are (1) the estimated resource usage of an application, (2) the regional locations of the current and future set of clients serviced by the application, and (3) the QoS-parameters of these client requests (bandwidth, delay, latency). A more detailed specification of these requirements can be found in the specification of the interface between the adaptive media server and Vagabond2.
3. Interfacing AMS and Vagabond2

Our intention when designing the interface between the adaptive multimedia server and Vagabond2 was to minimize the coupling between them. Some parts of the interface are implemented on the side of the media server, some on the side of Vagabond2. Either party only depends on the interface when communicating with the other one. There are three major parts of the interface definition (given in CORBA IDL syntax): basic, application-specific, and adaptation-specific.

The basic part holds interfaces that are implemented on the media server’s side. They provide information about the applications Vagabond2 should run, and the clients using these applications. The basic part is defined in module vagabond. **Date** defines a structure to hold data about a time instance. **Struct RequestInfo** contains detailed information about a client request (e.g. avg. bitrate, max. delay, max. latency). **Interface Client** defines the information to be provided about a client using one of the migratable/replicatable applications (client’s IP address, port number, …). **Interface ApplicationInfo** provides all required information for running and managing an application on a remote host (application’s name, constructor parameters, Jar-file, …).

The interfaces in the application specific part are implemented in Vagabond2. They provide services for loading, evacuating and locating applications. The application specific interfaces are defined in module vagabond::service::management::application. **Interface ApplicationLocator** provides means for locating all hosts on which a certain application currently runs on. Each application that has been loaded on at least one host has an own application locator. **Interface ApplicationService** provides means for loading and evacuating a certain application on a certain host, helps locating applications, and helps finding the applications on a given host. It is the center of the application part.

The interfaces of the adaptation specific part are implemented in Vagabond2. They provide services for adapting the applications loaded by Vagabond2 with respect to an optimal distribution. The interfaces and data structures of the adaptation specific part are defined in module vagabond::services::management::adaptation. **Interface HostRecommender** is responsible for recommending a set of hosts for a certain application by using topological (global) and resource usage (local) information. It should also check whether an application has optimal distribution. **Enum AdaptationPolicy** represents an adaptation policy to be used by the adaptation service, like e.g. load balanced, or minimum startup delay. **Interface AdaptationService** provides access to objects being responsible for and dealing with application-adaptational issues. It can return a host recommender, may suggest a policy by taking into account the data of clients, and can adapt an application by reorganizing its distribution.
Vagabond2 is based on Vagabond, a mobile agent system developed at the Budapest University of Technology and Economics [4]. It is a CORBA-based [9] mobile agent system written in Java and was originally developed to be used in multimedia database environments [5]. Later it became a good candidate as an infrastructure for an adaptive multimedia server. However, there were issues which made it necessary to modify Vagabond.

Vagabond2 is also CORBA-based and written in Java. It is able to migrate CORBA objects, and pass their references to their clients. It keeps track of the available server hosts, the applications they are running and the CORBA objects that incarnate the applications. These functionalities are hidden behind the interfaces described in section 3.

Two public interfaces are provided: AppService and AppLoc (“Application” is abbreviated to “App”, “Locator” to “Loc” in this section). The system will consist of server hosts that are able to load and run Java-CORBA objects, and applications that run on some of these servers. An application therefore consists of several CORBA objects on different hosts, where the objects can be thought of as ‘incarnations’ of the application.

The two major responsibilities are (1) providing the server hosts that can load Java objects and publish them as CORBA objects, and (2) maintaining the information of the relations among applications, their incarnations, and server hosts. The former required a simple modification of the Vagabond system.
We defined a new interface *Harbour*, which will take care of this responsibility. Concerning the second responsibility it is clear that there are one-to-many relations between an application and its incarnations, and between a host and the incarnations running on it. For the sake of efficiency we decided to introduce a set of redundant relations: \{application, hosts\}, \{application, incarnations\}, \{host, applications\}.

On figure 2 the classes and interfaces of Vagabond2 are shown. The *Harbour* CORBA interface defines the methods provided by objects that are able to receive, run, and evacuate Java objects, and connect them to the ORB. Every server host must run such an object.

The classes starting with “Host” are responsible for storing the information about available server hosts. A HostData object stores information about a single host: a reference to an object implementing the *Harbour* CORBA interface, and the names of the applications currently running on that host. HostStore manages \{hostname, HostData\} couples. HostService provides a CORBA interface for updating the HostStore remotely.

The AppData objects store the \{hostname, Object\} couples for a given application, where the Object is a CORBA object that was loaded on the given host. It also stores a reference to the AppInfo object that describes the given application. AppStore stores \{appname, AppData\} couples.

The AppService_impl implements the AppService CORBA interface, has access to the HostStore and AppStore objects, and has a list of objects implementing the AppLoc CORBA interface for every loaded application. It will load and evacuate the objects incarnating an application on Harbours that are registered by HostService. The load() and evacuate() methods (1) manage the information about newly loaded or removed objects in HostStore and AppStore; (2) call the load() and evacuate() methods of the Harbour object concerned.

5. **Conclusion and Future Work**

We addressed the requirements of the distributed adaptive multimedia server architecture AMS to its underlying infrastructure Vagabond2 and vice versa, regarding the management and adaptation of multimedia applications. From the point of the server the middleware has to provide two major classes of services, namely *application* and *adaptation* services. On the other hand, for Vagabond2 performing its tasks of managing applications and recommending hosts, the server has to provide detailed information on the applications to manage, as well as on the clients requesting services from those applications (i.e. information on location and request parameters). This set of bilateral requirements resulted in the specification of a CORBA-based interface between AMS and Vagabond2. Using this interface Vagabond2 provides means for (1) loading and starting CORBA objects written in Java on any host that runs Vagabond2,
(2) evacuating an object, and (3) querying the location and distribution of objects. All of these services can be accessed by CORBA object invocations.

In the near future Vagabond2 should also support the adaptation using the knowledge gained from the network topology, the load of links and hosts, and the needs of the clients. This knowledge should be used to implement different adaptation strategies in the host recommender. Furthermore, a complete implementation and detailed evaluation of a distributed adaptive multimedia streaming application based on MPEG-encoded media streams is planned, using Vagabond2 as the infrastructure for performing adaptational issues. In particular, it should be figured out which adaptation strategy performs best under a given set of constraints (e.g. the application distribution and the request parameters). Moreover, it should be investigated whether Vagabond2 would be able to perform on the fly migration or replication of server components, and under which restrictions this would be the case.

References