Content-based Indexing and Retrieval supported by Mobile Agent Technology

Harald Kosch, Mario Döller and László Böszörményi

Institute of Information Technology, University Klagenfurt, Austria
harald.kosch(mdoeller,laszlo.boeszoermenyi)@itec.uni-klu.ac.at

Abstract. In this paper we present the MultiMedia Database Mobile agent technology (M^3) which supports personalized content-retrieval and indexing in a distributed Oracle 8i DB. We implemented an agency on top of the Oracle 8i JServer and realized mobility with the embedded Visbroker Corba ORB. A performance comparison of our mobile agent technology with a client-server solution for a nearest-neighbor search in an image database shows the efficiency of the proposed solution.

Keywords: Mobile Agents, Multimedia Database System, Content-based Indexing and Retrieval

1 Introduction

The increasing development of indexing and retrieval tools in distributed multimedia database systems (MMDBMS), as well as the growing quantity of multimedia data, require efficient technologies to ensure the access and the management of the network and client resources [1].

In this context, we take advantage of mobile agent technology as an enhancement of distributed multimedia database indexing and retrieval. Mobile agents allow the execution of the retrieval tasks in an automated way, with minimal human interaction [2]. This allows the user to concentrate on other client activities, like the preparation of the client's buffer/cache for the expected multimedia delivery. Furthermore, it fits well with the requirement of efficient multimedia indexing and retrieval based on user's preferences by offering personalized processing of multimedia data through the access and pre-processing of the multimedia raw data where they are stored. For instance, a user is interested in retrieving the nearest images to a reference image in a multimedia database, however the database proposes only a simple range-search. Using the mobile agent technology the agent can contain a method, provided by the user, for processing the nearest-neighbor search directly in the database management system.

Internet applications, actually using effectively the mobile agent technology are for instance electronic commerce [3], telecommunication [4], information retrieval [5], and management of distributed resources [6]. This paper introduces a mobile agents technology in the scope of content-based indexing and retrieval in a distributed Oracle8i MMDBMS. We will demonstrate that these problems are well suited to mobile agent technology (see section 3). Yet, astonishing, that few related work considered mobile agent technology in this context (see section 2), although applications running on top of a MMDBMS have the very

similar features to the applications mentioned beforehand : they are characterized by asynchronous transactions, high latency, complex information processing and distributed task processing features.

2 Related Work

One key functionality in a multimedia database (MMDMBS) is how to index and then to retrieve continuous and non-continuous multimedia information efficiently. One broadly used method, the *Content-Based Retrieval (CBR)* of multimedia objects relies on extraction properties of a multimedia object [7, 8]. CBR in distributed MMDBMS involve the retrieval of multimedia data from various, possibly heterogenous database sites, and to compute the result in mutual agreement.

A typical approach to CBR is the similarity search between extracted multimedia features [7]. Here, the query is actually posed against the low-level feature vectors extracted from the multimedia object. A broadly used Image CBR system is `QBIC` (Query By Image Context, see wwwqbic.almaden.ibm.com), another popular system for videos is `Virage` (www.virage.com). Features described for CBR include measures expressing the color/texture/shape distribution of an image plus motion for a video. A CBR query is translated into a point query in a multi-dimensional feature space. The similarity between a query- and a database-object is estimated using a distance function.

CBR in a distributed MMDBMS is broadly supported by a client/server architecture. It includes (1) user interfaces to submit requests, their transfer to the DB server; (2) the retrieval operations at the DB server; and (3) the results return. A broadly used protocol for the client (written in Java) and server is JDBC (Java Database Connection; see java.sun.com/products/jdbc/index.html). If several sites are involved, multiple client-server connections are spawned and the different results are compared and merged at the client-side. An example is `MetaSEEk` (www.ctr.columbia.edu/MetaSEEk/). It is a meta-search engine for images based on the content of CBIRs located at IBM, Virage and Columbia University servers.

Our previously developed `SMOOTH` system [9] provides besides CBR also means for high-level annotation and querying, works in a client-server environment as well. A recent system enhancement (introduction of 'Domain Transparency' meaning that the client interface automatically adapts to extensions made by a new application domain to the base annotation classes) revealed serious performance bottlenecks of the JDBC connection (thin driver) to an Oracle 8i DB. The repeated JDBC calls to build the client interface dynamically combined with a high volume of requested data worsened the response time considerably. A first-aid solution was the integration of a JDBC client cache for query results. However, the use of a mobile agent solution might improve the situation more efficiently. We address such a solution in the near future.
Many mobile agent systems have been established. Some of these are Aglets [10], Mole [11] and Grasshoper [12]. They are well-suited to a wide range of Internet applications, to mention only WWW mining [13], or telecommunication [14].

However, to the best of our knowledge, there are few works which deal with the use of mobile agents in a distributed database system. Some related work concentrated on the use of mobile agents in distributed Web databases [15, 16]. Others dealt with distributed data warehousing. For instance Weippl et al. [17] propose a mobile AgentDB technology, based on the JServer capabilities of the Oracle 8i DBMS which is tailored to the inserting process in a data warehouse. These related work rely mainly only on simple access functions and do not address the problem of handling multimedia data. In the context of distributed multimedia systems, mobile agents are successfully utilized for Quality of Service (QoS) negotiations [18, 19]. For instance, Manvi et al. [19] propose a mobile agent based QoS management system which provides means for efficient agent based bandwidth negotiation. However, the issue of CBR is not yet treated in this context.

3 Mobile Agent Technology in a Distributed MM-System

3.1 General Considerations

A mobile agent is defined as a self-contained software element that acts autonomously on behalf of a user (e.g. person or organization, or a multimedia content customer) and in addition, has the unique ability to migrate from one host in a network to another [16].

The definition of a mobile agent contains at least three issues which deserve special interest when developing mobile agent systems in a distributed multimedia database systems. The first issue refers to the autonomy of a mobile agent. This is a feature that allows the agent to act on its own by using the data (e.g. the feature vector of an image whose nearest-neighbors have to be searched), and the mobile logic which it incorporates (e.g. the similarity search code), and requires only little human intervention (e.g. provide the iterinary) or guidance (e.g. error handling). Further, the time needed to fulfill the tasks is reduced because interaction with the user is avoided. For instance, a similar image to the reference one found in the first database is used as input for the search in a second image database. Obviously, the agent has to be designed to deal with any situation that may occur during execution, such as the violation of the private information associated to the agent's owner [20], or a resource violation at the remote agency which let return the agent immediately.

The second issue in the definition refers to the mobility of an agent. The mobility feature enables the agent to travel to the host where the data are physically stored. In this way the transfer of a large amount of data in the network is prevented. This is obviously of great interest in a distributed multimedia database system. For instance reconsider a distributed nearest-neighbor search. Instead of downloading the best matches from all involved databases, as required in a client/server solution, the mobile agent incorporates the result of the first visit as internal state and uses it for the further search.

The final argument for using mobile agent technology is the personalization of the indexing and retrieval process. The effectiveness of automatic multimedia indexing is conditioned to a great extent by the amount of a priori information

available in the application domain [21]. An elegant solution is the integration of this information as mobile logic. For instance, in an surveillance application a mobile agent might include detailed information on the intruder objects to the video server which allows the selective coding of video clips (e.g. as MPEG-4 Video Objects) [22].

3.2 Architecture of our M^3 Agency

In the following we describe a set of concepts upon which our mobile agents M^3 system relies. Principally, it is a Java-based mobile agent system which uses CORBA services to implement mobility. Its main characteristics are the agent execution in the core of the database, i.e. the agency and the agent are database objects, and the possible use of direct access and retrieval methods of multimedia data through server-sided JDBC and *inter*Media Java classes (see section 3.2).

Every Oracle 8i MMDBMS server which wants to host an M^3 agent must provide an agent *run-time environment*. This environment is responsible for execution and migration of the agents. Furthermore, an *agent dispatcher* is needed to initially start up the run-time environment. The role of the agent dispatcher in an Oracle DB is played by the JServer. The JServer offers an encapsulated environment, sessions, which are independent from each other, as well the CORBA advanced services. Our run-time environment relies on the offered CORBA services, as well as on the Java language features, to implement mobility. The advantages of this approach are manifold, e.g. use of security mechanism offered by CORBA services, use of the advanced naming service to locate the MMDBMS servers, as well as use of object serialization for conversion and reconstruction of java classes and instances, in connection with the Java networking support. Similar strategies have been successfully employed in related systems (see [23] for CORBA services, and [24] for Java-based mobile agents).

Introduction to Oracle JServer The architecture of the JVM in Oracle 8i is based on a session model. Sessions are private address spaces that clients have exclusive access to. They are not shared with any other client, although they may be serially re-used between clients. This means that the actions of one client can not interfere with any other client. All interactions between clients is done through the database itself, using transactional semantics. Each session has its own JVM. It is a 'thin' JVM which uses the shared memory architecture. This means that all clients share the read only static portion of their own JVM.

Java classes and its sources in Oracle 8i are not stored in the file system, but in the database system the same way that PL/SQL packages are stored. Java classes, sources and resources are therefore database objects. Java Threads running in the Oracle 8i session are scheduled non-preemptively, i.e. a thread must yield control explicitely in order that the JVM runs another thread.

The whole environment is called JServer. Furthermore, the JServer consists of a Java Accelerator, an integrated CORBA 2.0 ORB (Visibroker), an embedded SQLJ translator and an Enterprise JavaBeans 1.0 compliant container for EJB components. The CORBA environment provides an user with the ability to call into and out of the database server by using the CORBA IIOP protocol. CORBA

servers can be invoked from clients using IIOP. In this scenario the database then behaves as a CORBA server.

Session Architecture in the M^3 Agency The JServer is responsible for the session management (creation, destruction etc.). Each session possesses a 'thin' JVM (˜40 KB), a session memory, which is used for static variables, and a call memory for the instance variables of the currently executed class. The session's memory is limited to secure the system and to offer some kind of equality between all sessions. Figure 1 shows the session architecture. Before a database object can be invoked (e.g. a java CORBA Server), all class files and all help files have to be loaded and to be "published". Agency and agent run in the same session. However, only the agency is published and activated via the Java Naming and Directory Interface (JNDI). The arriving mobile agent is received by the agency and loaded with a Class Loader provided by the JServer. The agency obtains then a reference to the agent and can interfere so at any time with it (stop, resume etc.). For the publishing of the agency, one needs to know the valid username and password of the user which hosts the mobile agents (agency publishing can only be done by an agency administrator). However, the agent itself needs not to know user/password, as the session is already spawn by the agency. Furthermore, the agent accesses the stored multimedia data via server-sided JDBC directly, without supplemental authentification (see subsection 3.2). Therefore, it is the task of the agency to provide the necessary security level for the agent execution (see next subsection).

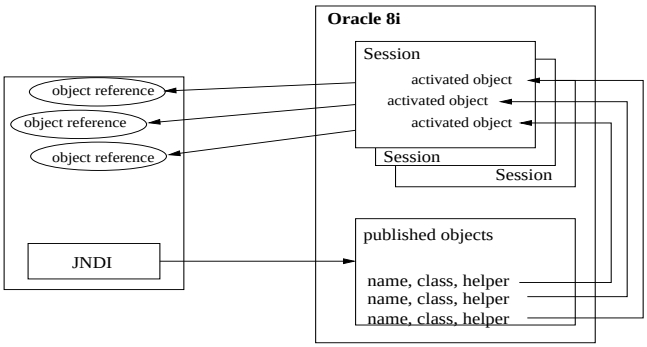

Fig. 1. *Session Architecture.*

Run-time Environment The M^3 Agent system is not an always running server, but is activated on demand. It reminds therefore of the servlet technology where for each call a new servlet instance will be created. In the following, we will give a short description of the necessary classes, database tables and database users:

– *AgencyServant*: This class is the published object in the database. Oracle's JServer will return a reference to a newly created AgencyServant in a new session to the calling client. Furthermore, the AgencyServant accepts the incoming agents and starts them in an AgencyThread. When an agent terminates, it will be sent to the next host in the itinerary. The AgencyServant

is also responsible for keeping track of the currently running agents. If a new agent arrives, the agency has first to test with the help of the `Agencyinfo` table whether the maximal number of running agents is reached.
- *AgencyThread*: Every agent is started in such a thread. The agency is able to start and stop this thread whenever it wants.
- *Migrateable*: Each agent has to implement this interface.
- *Agent_Sec database user*: This is the security user of the agency. This user has system privileges and can grant or revoke rights to our agency user. Every access to the file system or tables of other users have to be granted by this user. Only the agency can contact the security user.
- *Itinerary*: This interface defines the methods of an itinerary. The agency, where the agent resides at the moment, can get the next target host name and is able to send the agent to the new agency.
- *XNotified*: This interface is used to communicate with the original client. Its implementation is usually on the original client side.
- *Agency database user*: This is the database user account where the agency is published. It has to be a user who only has access to his own database space. No further rights are necessary. We use SCOTT in our current test environment.
- `Agencyinfo` database table: The `Agencyinfo` table contains information of the agency and the currently running agents. The table consists of the following entries:
  - ACTIVE: contains the number of currently running agents in the database.
  - MAX: maximal number of simultaneously running agents.
  - SERVERNAME: The DNS-name of the database server.
  - DATABASENAME: The SID of the database.
  - USERNAME: Name of the agency user. In this way the agency is able to run in different user spaces on different databases. However, the security user needs to know whom to grant and restrict rights.

Migration in the M^3 Agency The migration of an agent occurs through the following steps:

1. The client creates an agent for his/her needs, serializes it and creates a jar-file with the class files.
2. The client creates instances of the implementations of the XNotified and the Itinerary interface and connects them to the ORB. The Itinerary contains a list of the servers which the agent intend to visit.
3. The reference of the first agency is returned by the Oracle's JServer handling the client call. The client sends then a message, containing the byte-code of the agent (as a jar-file-type byte array), the serialized state of the agent, the Itinerary and a handle (remote reference) to the Xnotified.
4. The agency receives the message and tries to create the agent. An agent can only be created iff the number of currently running agents is lower than the maximal number of running agents specified in the `Agencyinfo` table. If there is a free slot for the new agent, then the value of the ACTIVE column

in the `Agencyinfo` table is increased by one. If the maximal number of agents is currently reached, then the agency makes three more attempts to get a free slot. If these attempts fail again, the agent is sent to the next host.

5. The agency loads the agent's classes into the database.
6. The agency creates an AgentThread and transfers the serialized data of the agent to the thread.
7. The thread deserializes the agent and calls its execute method.
8. When the agent and the AgentThread terminates (no more host in the Itinerary), the agent's result will be sent back to the Xnotified object of the client.

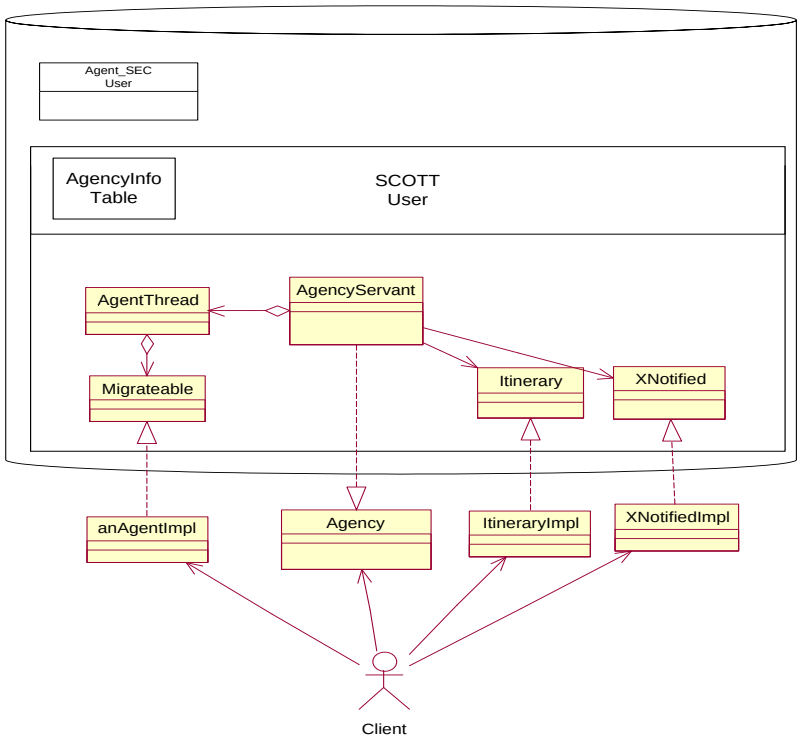

Fig. 2. *Architecture of the M^3 Agency System.*

9. The agency serializes the agent, "gzips" the resulting stream and contact the next host in the itinerary in order to prepare the sending process. If the next host is accessible, the agent is sent to the next host and the procedure pursues through step 4.
10. After having sent the agent, the agency cleans up the system and decreases the value of the ACTIVE column in the `Agencyinfo` table.

If an exception occurs during the execution of the agent, and it is not caught by the agent, the thread saves the exception. The agency gets then this exception and returns it to the Xnotified object.

Support for CBR in the the M^3 Agency A mobile agent in M^3 may build up personalized and efficient CBR through two mechanisms, which are related strongly to each other. First, through the possibility of server-sided JDBC for a direct access to database objects and second through the use of *inter*Media Java classes for the access, retrieval and streaming of multimedia data (for more information on *inter*Media see technet.oracle.com). The use of both mechanisms are enabled by the implementation of the agency as database objects and the execution of the agent inside a database session.

The Oracle JDBC server-side internal driver (KPRB) is built into the Oracle JServer and is intrinsically tied to the Oracle8i database and to the JVM. The driver runs within the default session–the same session in which the JVM was invoked. It is optimized to provide direct access to SQL data and PL/SQL subprograms on the local database, through native function call (no Net8 call involved). Running the mobile agent inside a database session has the advantage that the mobile agent needs not to provide the database name, the agency user and password for obtaining a database connection through server-side internal driver. It is the task of the agency to spawn the session which hosts the agent and to load its byte-code, as well as its instance directly into the database. Therefore the same agent code can be executed without no adaptation on any host participating in the distributed MMDBMS system.

The server-sided internal driver is one prerequisite for the support of CBR, the other one are the *inter*Media Java classes provided by Oracle 8i. Oracle8i *inter*Media supports multimedia domain-specific types, methods, and interfaces. The following multimedia object types are supported: ORDAudio for audio, ORDImage and ORDVir for images, and ORDVideo for videos. For all of these data types, special *inter*Media Java classes are provided which enables the user to create own Java applications to use, manipulate, and modify multimedia data stored in an Oracle 8i database.

The multimedia retrieval part of the mobile agents has to be designed in the following way, first make a server-sided connection from the Java application to the Oracle database through JDBC's `defaultConnection()` method (no user/password needs to be provided). Second, execute a SELECT statement on the database table containing multimedia data and store the results in the Java application (based on the *inter*Media Java classes). Third, move the results into an *inter*Media object with the `getCustomDatum()` method. Perform operations on the Java multimedia application object to obtain the desired functionality (possibly through repeated SELECT statements).

Example: The following statements show how to retrieve the first stamp of an *image table* `stamps` (1), how to move the results into an *inter*Media object `imgObjj` (2), and finally how to produce an *MemoryImageSource* `mis` for further processing. This *MemoryImageSource* can e.g. be used for the computation of a color histogram (see the mobile agents in the experimenal section 4).

```
...
(1) Statement stmt = conn.createStatement();
(1) OracleResultSet rs  = (OracleResultSet)stmt.executeQuery
          ("SELECT image1 FROM stamps WHERE id=1");
```

```
(2) OrdVir imgObjj = (OrdVir)rs.getCustomDatum(1, OrdVir.getFactory());
(3) int width  = imgObjj.getWidth();
(3) int height =  imgObjj.getHeight();
(3) MemoryImageSource mis =
          new MemoryImageSource(width, height,ColorModel.getRGBdefault(),
          imgObjj.getDataInByteArray(), 0, width);
...
```

4 Performance Evaluation

The performance evaluation is given in two parts. In a first part we show how our agent system might enhance the server capabilities for the example scenario of a nearest-neighbor search in a stamp database. In a second part, we compare for the same problem, but for a larger face database, the response time of our agent system to a solution based on a client-server architecture.

4.1 Evaluation of the Enhancement Capabilities

One of the main advantages of using our mobile agent systems concerns its capacity to enhance the database functionality by injecting personalized programs in the database system.

Fig. 3. *Reference Image of the NN-search*

The example image database contains 59 US stamps in jpeg format, each with 24 bits per pixel (16 Million colors). The example problem to be solved within this part is that of a nearest-neighbor search (NN-search). That means we detect the most similar stamp to a reference one, in the stamp image database. Such a search is useful for many applications, for instance a user finds an interesting stamp in a newspaper, scans it and likes to retrieve more information (e.g. price, where to obtain, etc.) about this stamp from a stamp database. Therefore, he/she intends first, to find the most similar stamp in the database and second, retrieve information as provided by it.

The indexing and retrieval task is solved by two agents, the first one (indexing agent) computes a color histogram feature vector of length 256 from the image database and stores it in a separate table. This table has two attributes, the feature vector of type `VARRAY(256) OF NUMBER`, and the id of the respective image. We suppose that the images are stored in an object-relational image table with

one attribute of type `ORDVir` referencing the images and one attribute of type `NUMBER` containing the id of the image stored. The second mobile agent (retrieval agent) performs the NN-search of the reference stamp (Aircraft 'Staggerwing') shown in figure 3. This stamp is not contained directly in the database, however appears as a thumbnail in a collection of images showing classic American Aircrafts (this collection is shown to the left of figure 4).

Fig. 4. *Left : Closest Image found by the Mobile Agent. Right : Images found by the* `OrdVir` *System for a Threshold Value of 26.*

Let us now compare our mobile agent to the build-in capacities of the Oracle8i *inter*Media Visual Information Retrieval system `ORDVir`. Please refer to technet.oracle.com for more technical information on `ORDVir`.

The `ORDVir` system provides CBR functionality through the object type `ORDVir` and associated methods and functions. CBR functionality is provided by the ability to extract an image feature vector from four different visual attributes : global and local color[1], texture and shape. The image comparison mechanism is provided by a similarity function `ORDSYS.VIRSimilar()` which takes two elements of the `ORDVir` type as input. Furthermore, the user must define a threshold similarity value governing the difference of the respective feature vectors, i.e. if the weighted sum of the distances for the visual attributes is less than or equal to the threshold, the function returns true, otherwise it returns false. Note that the same principle is also applied in other systems, e.g. the Informix Excalibur Image DataBlade module (see examples.informix.com and then goto Using DataBlade modules).

Experimental Protocol

a) M^3 Agent System: The mobile agent incorporates the feature vector of the reference image. After the complete scan of the stamps database it returned the closest image to the reference one, as shown to the left of figure 4.

b) `ORDVir` System: In order to implement the NN-search with the `ORDVir` system, one has to know exactly the threshold value. We started to use a value (10) of

[1] Global color represents the distribution of colors within the entire image. Local color represents color distributions and where they occur in an image.

the reference examples provided in Oradoc[2]. Furthermore, in order to compare the two algorithms on the same feature extraction base, we specified that the globalcolor visual attribute has to be used exclusively. With this first threshold value (10), the `ORDVir` system didn't find any matching image. We augmented then the value to 30, and then the system found 5 matching images. For the values 29 and 28 we found three matching images and for a value of 26 we found 2 matching images (they are shown in the middle and to the right of figure 4). Then for a value 25 we found the closest image, as shown to the left of figure 4. Finally, beyond the threshold value of 25 no match can be returned.

Obviously, such a search for an adequate threshold value is not acceptable for a NN-search (multiple scans of the image table are required) and our mobile agent represents therefore a useful enhancement to the `ORDVir` functionality for the retrieval tasks to be solved.

4.2 Response Time Evaluation

Here, the NN-search of the qualitative first part is adapted for a response time evaluation through considering a larger face database of size 4000. We are interested in computing the nearest-neighbor image of a reference image in the face database. Thereby, we merged the functionality of the former index- and retrieval agent into one NN-search agent. The sample database used was the AR Face Database created by Aleix Martinez and Robert Benavente at the Computer Vision Center (CVC) at Purdue University. It contains 4000 images corresponding to 126 people's faces (70 men and 56 women) [25].

The testing environment consisted of two similar agent hosting machines with the following parameters, CPU : AMD 800 MHz CPU, OS : Win NT 40 SP6 and Program Environment : JDK 1.3. Both machines communicated over a 100 MBit/sec Ethernet segment. Furthermore on each machine the Oracle 8.1.7 database was installed.

Our mobile agent computes on the database server for each face first a feature vector with 256 values. Second, it compares the difference of this vector and the reference vector to the difference of the yet best match to the reference one (either found on a previous host or on the same host). If the difference is smaller than the previously found best one, it retains it for further processing. The client-server solution has to download each image to the client, and performs the image comparison locally.

The metric, we examine here, is the response time for computing the nearest-neighbor for the client-server solution and the mobile agent solutions. The performance of the mobile agent solution depends on the database load, which depends itself to a great extent on the available free main memory. To account for this, we limited the available server java pool memory space for each session, once to 20 MB, 60 MB and once to 100 MB. The number of faces in the databases was varied from 400 to 10400 (database size). For a size n smaller than 4000, the first n images of the available ones are retained, for a size greater than 4000, the images are replicated. We performed two experiments: in the first we used one host and in the second two hosts.

[2] www.oradoc.com/ora817/inter.817/a85333/virj_ref.htm

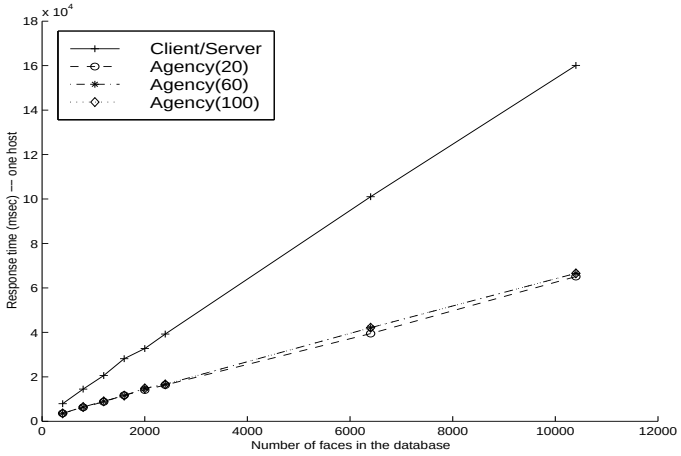

Fig. 5. *Response Time of the Client/Server Solution vs. our Mobile Agent for one Host.*

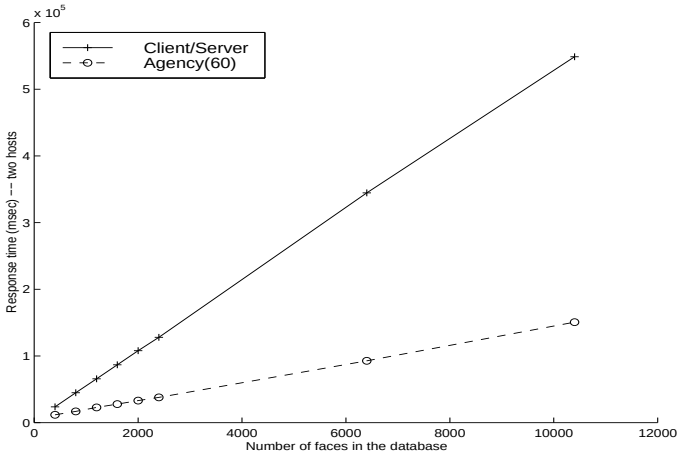

Fig. 6. *Response Time of the Client/Server Solution vs. our Mobile Agent for 2 Hosts.*

Figure 5 shows the response time of a first experiment using one host depending on the available java pool size and the database size. The figure clearly shows that our mobile agent solution outperforms the client-server case. The response time of the client-server solution is on average 2.38 times higher than that of the mobile agent one. The response time of the mobile agent solution is almost invariable to the size of the available java pool size per session and varies only minimally between the different values, e.g. the highest variance of 2.5% occurs between 20MB and 100MB.

Figure 6 shows the response time of a second experiment using two hosts depending on the available java pool size and the database size. Once again, the response time of the mobile agent solution is clearly smaller than that of the client-server solution. Moreover the performance gain (3.08 times less response time of the mobile agent) is higher than in the first experiment (2.38 less response time). The response time is again invariable with respect to the java pool size

and therefore not shown in figure 6. If one compares now the results using two hosts to the results using one host, it can be noticed that the response time of the mobile agent solutions doubles approximately compared to the first experiment, as the response time of the client-server solution is more than two and half times higher than that of the first experiment. This is due to the intermediate processing on the client machine in the client-server solution. Therefore, the mobile agent solution scales better with respect to the number of hosts involved in the search.

5 Conclusion

One of the main advantages of the mobile agent technique is the ability of migration and mobility, that means to bring the problem solver directly to the problem which results in processing cost and time benefits. This paper shows that it is possible and profitable to establish an agency system inside a distributed Oracle 8i multimedia database.

In this context, we implemented the M^3 -**M**ulti**M**edia Database **M**obile agents- which supports personalized content-indexing and retrieval (CBR) in a distributed Oracle 8i multimedia database system. Support for CBR follows directly from the implementation of the agency system inside the Oracle 8i database (i.e. both agency, as well as agent are database object) and by using server-sided JDBC combined with the *inter*Media Java Class libraries. Our agent system proposes an advanced security concept, through session management and an own security user which grants and revoke database and resource access rights for the agent. A final performance comparison of our mobile agent technology with a client-server solution for a nearest-neighbor search in an image database shows the efficiency of the proposed solution.

In the near future we will rely the implementation of our multimedia database SMOOTH [9] on the M^3 mobile agency, in order to overcome the communication bottleneck of a client/server JDBC (see section 2).

References

1. Guojun Lu. *Multimedia Database Management Systems.* Artech House, 1999.
2. N.M. Karnik and A.R. Tripathi. Design issues in mobile-agent programming systems. *IEEE Concurrency*, 6(3):52–61, July/September 1998.
3. P. Dasgupta, L. E. Moser, and P. M. Melliar-Smith. The security architecture for MAgNET: A mobile agent E-commerce system. In *Third International Conference on Telecommunications and E-commerce*, Dallas, TX, USA, 2000.
4. H-Jeon, C. Petrie, and M.R. Cutkosky. JATLite: A Java agent infrastructure with message routing. *IEEE Internet Computing*, 4(2), March/April 2000.
5. B. Brewington, R. Gray, K. Moizumi, D. Kotz, G. Cybenko, and D. Rus. Mobile agents in distributed information retrieval. In *Intelligent Information Agents*, chapter 12. Springer Verlag, 1999.
6. P. Bellavista, A. Corradi, and C. Stefanelli. Mobile agent middleware for mobile computing. *IEEE Computer*, 34(3):73–81, March 2001.
7. Y. Rui, T. S. Huang, and S.-F. Chang. Image retrieval: Past, present and future. *Journal of Visual Communication and Image Representation*, 10:1–23, 1999.

8. A. Yoshitaka and T. Ichikawa. A survey on content-based retrieval for multimedia databases. *IEEE Transactions on Knowledge and Data Engineering*, 11(1):81–93, 1999.
9. H. Kosch, R. Tusch, L. Böszörményi, A. Bachlechner, B. Dörflinger, C. Hofbauer, C. Riedler, M. Lang, and C. Hanin. SMOOTH - A distributed multimedia database system. In *Proceedings of the International VLDB Conference*, Rome, Italy, September 2001. Accepted for Publication as Demonstration Paper.
10. D.B. Lange and M. Oshima. *Programming and deploying Java mobile agents with Aglets*. Addison-Wesley, Reading, MA, USA, 1999.
11. J. Baumann, F. Hohl, K. Rothermel, and M. Strasser. Mole - Concepts of a mobile agent system. *World Wide Web*, 1(3):123–137, 1998.
12. C. Bäumer, M. Breugst, S. Choy, and T. Magedanz. Grasshopper — A universal agent platform based on OMG MASIF and FIPA standards. In *First International Workshop on Mobile Agents for Telecommunication Applications (MATA'99)*, pages 1–18, Ottawa, Canada, October 1999. World Scientific.
13. H. Ouahid and A. Karmouch. An XML based web mining agent. In *First International Workshop on Mobile Agents for Telecommunication Applications (MATA'99)*, pages 393–404, Ottawa, Canada, October 1999. World Scientific.
14. C. Bäumer and T. Magedanz. Grasshopper : A mobile agent platform for active telecommunication networks. In *Proceedings of the 3rd International Workshop on Intelligent Agents for Telecommunication Applications (IATA-99)*, pages 19–32, Berlin, Germany, August 9–10 1999. LNCS 1699, Springer Verlag.
15. S. Papastavrou, G. Samaras, and E. Pitoura. Mobile agents for WWW distributed database access. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 228–237, Sydney, Australia, March 1999.
16. Dejan Milojicic. Mobile agent applications. *IEEE Concurrency*, 7(3):80–90, July/ September 1999.
17. E. Weippl, J. Altmann, and W. Essmayr. QoS management by mobile agents in multimedia communication. In *Proceedings of the International DEXA'2000 Workshops*, pages 477–481, Greenwich, UK, September 2000.
18. L.A. Guedes, P.G. Oliveres, L.F. Paina, and E. Cordozo. An agent based approach for supporting quality of service. *Computer Communications*, 21:1269–1278, 1998.
19. S. Manvi and P. Venkataram. QoS management by mobile agents in multimedia communication. In IEEE CS Press, editor, *Proceedings of the International DEXA 2000 Workshops*, pages 407–411, Greenwich, London, UK, September 2000.
20. C. Tschudin. Mobile agent security. In *Intelligent Information Agents: Cooperative, Rational and Adaptive Information Gathering on the Internet*, pages 431–445. Springer Verlag, 1999.
21. S.-C. Chen, R.L. Kashyap, and A. Ghafoor. *Semantic Models for Multimedia Database Searching and Browsing*. Kluwer, 2000.
22. P. Correia and F. Pereira. The role of analysis in content based video coding and indexing. *Signal Processing*, 66(2):125–142, 1998.
23. M. Amer, A. Karmouch, and T. Gray. Adding mobility to CORBA. In *First International Workshop on Mobile Agents for Telecommunication Applications (MATA'99)*, pages 143–160, Ottawa, Canada, October 1999. World Scientific.
24. D. Wong, N. Paciorek, and D. Moore. Java-based mobile agents. *Communications of the ACM*, 42(3):92–102, February 1999.
25. A.M. Martinez and R. Benavente. The AR face database. Technical Report CVC Technical Report Number 24, 1998, Computer Vision Center (CVC) at Purdue University, 1998.