Sharing Mobile Databases in Dynamically Configurable Environments

Angelo Brayner and José Aguiar M. Filho

University of Fortaleza (UNIFOR), Av. Washington Soares, 1321
60811-341 Fortaleza, Ceara, Brazil
{brayner, jaguiar.mia}@unifor.br

Abstract. In an environment with support for mobile computing, we may have a collection of autonomous, distributed, heterogeneous and mobile databases, denoted Mobile Database Community (MDBC), in which each database user can access databases in the community through a wireless communication infrastructure. In such an environment, new participants may join to an MDBC as they move within communication range of one or more hosts which are members of the MDBC. Furthermore, MDBC participants may transiently disconnect from the network due to communication disruptions or to save power. Therefore, an MDBC can be characterized as a dynamically configurable environment. This paper describes an agent-based architecture, denoted AMDB (Accessing Mobile Databases), which enables such communities to be formed opportunistically over mobile database hosts in ad hoc configurable environments. The AMDB architecture is fully distributed and has the capability of exploiting physical mobility of hosts and logical mobility of database queries and their results across mobile hosts.

1 Introduction

Advances in portable computing devices and wireless network technology have led to the development of a new computing paradigm, called mobile computing. This new paradigm supports that users carrying portable devices are able to access services provided through a wireless communication infrastructure, regardless of their physical location or movement patterns. Mobile computing paradigm has affected traditional models and concepts of many computer science areas. For example, in the network area, networks need to be ubiquitous, since they must guarantee user connectivity independently of the physical user location. Communication networks with such as property are called *ad hoc* networks [12]. With respect to software engineering, that new paradigm has introduced the notion of mobile code, which refers to the capability of dynamically changing the bindings between code fragments and the location where they are executed. Moreover, a mobile code needs to be "aware" about the computational environment on which it is running [10].

The database technology has also been impacted by the mobile computing paradigm. For example, in an environment with support for mobile computing, a varying number of mobile computers can be interconnected through a wireless communication infrastructure, on each of which resides a database system. In other words, a dynamic collection of autonomous mobile databases, interconnected through a wireless

J. Eder and M. Missikoff (Eds.): CAiSE 2003, LNCS 2681, pp. 724–737, 2003.

communication infrastructure, can be formed in such an environment. For now on, that collection of mobile databases will be denoted mobile database community (MDBC). Databases in an MDBC are mobile, autonomous and distributed. Moreover, they can be heterogeneous.

Therefore, sharing information among multiple heterogeneous, autonomous, distributed and mobile data sources has emerged as a strategic requirement which should be supported by the database technology. Mobility allows any time and any place data access. Accordingly, conventional concepts and techniques, used in areas, such as query processing, transaction management and data distribution, should be revisited in order to introduce the concept of mobility into the database technology. In order to illustrate this fact, consider a mobile database community M. Suppose that databases belonging to M are heterogeneous and reside in mobile computers interconnected through an ad hoc network. Now, consider a database DB which resides on a mobile computer C and it is member of M. Thus, the data of DB should be shared with the other members of the community. For this purpose, the computer C should behave as a database server belonging to M. Additionally, a user on the mobile host C may wish to query objects stored at other members of M. Of course, the heterogeneous, distributed, autonomous and mobile databases belonging to the community have to be integrated. Since the databases are mobile, a given database DB can leave the community at anytime, even though a query submitted by DB is still running in other database systems in M. Moreover, the query submitted by a user on the mobile host C can represent a distributed transaction. In this case, a protocol to guarantee the atomicity of the commit operation [1] is required. Consequently, it is necessary to address the following problems in order to make feasible the access to mobile databases in an MDBC: *(i)* heterogeneous databases integration; *(ii)* query processing over a variable number of mobile databases and; *(iii)* mobile transaction management (i.e., concurrency control and recovery for mobile and distributed transactions).

In this paper we propose an architecture, denoted Accessing Mobile Database (AMBD), for supporting database sharing in mobile database communities. The key goal of the proposed architecture is to provide access to heterogeneous, autonomous and mobile databases. The AMDB architecture is based on the concept of mobile agents and is fully distributed. Furthermore, the proposed architecture provides the necessary support for forming MDBCs opportunistically over a collection of mobile databases hosts.

It is important to note that the AMDB architecture does not require changes to the core of underlying databases systems. For that reason, we classify the AMDB approach as non-intrusive [6]. This property enables commercial DBMSs for participating in a mobile database community through the AMDB architecture. Several approaches to support mobility should be classified as intrusive, since they require that new capabilities and functionalities have to be added to the underlying database [5,7].

This paper is structured as follows. In section 2, a mobile computing environment is characterized. In section 3, the concept of mobile databases community is defined and proposed architecture is presented and analyzed. Section 4 addresses the query processing mechanism for the AMDB architecture. Mobile transaction processing issues in the MDBC context are addressed in Section 5. Section 6 concludes the paper.

2 A Mobile Computing Environment Model

In a mobile computing environment, mobile computers are grouped into components denoted cells. Each cell represents a geographical region covered by a wireless communication infrastructure. Such cells can represent a Wireless Local Area Network (WLAN), an ad hoc network, an geographical area covered by a cell phone network (called cell as well) or a combination of those communication technologies (for example, an ad hoc network inside cell of a cellular phone network).

Mobile support stations (or base stations) are components of a mobile computing platform which have a wireless interface in order to enable communication between mobile units located at different cells. Thus, each cell must be associated with one mobile support station. Communication between two base stations is made through a fixed network. In fact, mobile support stations represent fixed hosts interconnect through a high-speed wired network. From a database technology standpoint, there are two classes of database systems in a mobile computing environment: *(i)* a class consisting of database systems which reside on fixed hosts, and; *(ii)* a class of database systems which reside on mobile computers. Database systems residing on mobile hosts are denoted mobile databases

It is worthwhile to note that, when a cell in a mobile computing environment represents a WLAN or an ad hoc network, mobile computers inside the cell can communicate with each other directly. On the other hand, when a cell represents an area covered by a cell phone network, mobile units need should use the cell's base station to communicate with each other. A mobile computer can use different communication technologies. We call *vertical handoff* the fact of a mobile computer migrating from a given communication technology to another. For example a mobile computer A can use a WLAN to communicate with other mobile computer inside the same cell, and it can use a cellular phone network to communicate with mobile computers located at another cell.

From a database technology perspective, we can categorize mobility in two different types:

i. *Physical Mobility*: this type of mobility cope with spatial mobility of database clients and database servers through different space regions;
ii. *Logical Mobility*: this type of mobility is related to code migration among several mobile clients and database servers. In order to provide logical mobility, mobile computers should be able to generate database access codes (SQL expressions, stored procedures or methods), which can migrate autonomously to several databases servers. When such a code arrives at a database server, it is locally executed. After that, it returns (with the access result) to home (the host which originates the access code).

It is important to note that the AMDB architecture provides support to both of the mobility types described above.

3 Sharing Databases in Mobile Database Communities

The key goal of the proposed architecture is to enable mobile database communities to be formed opportunistically over mobile database residing on mobile hosts in envi-

ronments with a wireless communication infrastructure. The proposed architecture is fully distributed (i.e. no centralized control structure is needed) and relies on existing mobile agent technology to allow transient sharing of databases. In addition, to support disconnected operations, the architecture exploits logical mobility of both database queries and their results. For the sake of clarity, we assume that mobile computers are interconnected through an ad hoc network infrastructure.

3.1 Mobile Database Communities

We characterize a database system as a mobile database, if the database system resides on a mobile unit. A Mobile Database Community (MDBC) represents a dynamic collection of autonomous mobile databases, interconnected through a wireless communication infrastructure. Observe that the notion of MDBC models the notion of a federation of databases which reside on mobile units.

A varying number of mobile computers can participate in an MDBC. In this case, MDBC members share databases available in the MDBC. The members of an MDBC can be categorized in two different classes: database server or database client. By database server, we mean that a mobile computer, which is the host of a database system (DBS). Each database system encompasses a database (DB) and a database management system (DBMS). A database represents a collection of object representing real world entities. A DBMS is the software component of the DBS. In that case, the database (or part of it) stored in a database server can be accessed by any other member of the community. The second class of MDBC members is composed by mobile hosts which participate in an MDBC as database clients. Such members can only access object stored in databases available in the MDBC. That means, either they do not have or they do not want to share an own database with the members of the community.

New participants may join to an MDBC as they move within communication range of one or more hosts which are members of the MDBC. On other hand, MDBC participants may transiently disconnect from the network due to communication disruptions or to save power. For that reason, an MDBC can be characterized as a dynamically configurable environment.

A mobile computer needs to declare to the participants of an MDBC its intention to become a member of the community. If the mobile computer is the host of a database system, it should declare that its database will be available to the members of the community. In that case, the mobile computer has to publish its database schema to the community.

It is highly likely that databases in an MDBC are heterogeneous. For example, they might be relational, native XML, oriented-object databases or any other data source (e.g., HTML pages). Thus, schemas of databases belonging to an MDBC should be represented in a common model. Since XML has been consolidated as a standard for data interchanging, we have decided to use XML as the common model for data interchanging in an MDBC. For that reason, mobile databases in an MDBC have to publish their schemas (or part of them) through XML Schema [23], a schema definition language for XML data.

Mobile users interact with mobile databases in an MDBC by means of transactions. A transaction models a sequence of operations on database objects that is executed atomically. Typically, such operations are expressed in a database query language

(e.g. SQL). There are two types of transactions in an MDBC environment: local transactions and global transactions. A local transaction is submitted directly to a mobile database DB by an application running on the same host on which DB resides. A global transaction consists of database operations which should be executed on different hosts belonging to an MDBC. The notion of global transaction models distributed transactions which are composed by mobile sub-transactions. Therefore, the notion of global transaction in an MDBC generalizes the concept of distributed transaction. Since a global transaction is distributed in different mobile hosts, a protocol should enforce commit atomicity of global transactions. An example for such a protocol is the two-phase commit (2PC) protocol [1].

The correctness criterion for the execution of concurrent transactions is serializability. We assume that each mobile database in an MDBC guarantees serializability by the two-phase locking (2PL) protocol [1]. This is a quite reasonable assumption, since all existing database systems implements the 2PL to guarantee serializability.

3.2 AMDB Architecture

The AMDB architecture is based on the concept of software agent. The proposed architecture is composed by two classes of agents: stationary agents and mobile agents. The stationary-agent class is compound by two types of agents: manager agents and wrapper agents. Manager agents are responsible for managing local computational resources of a mobile computer. These resources can be used to perform tasks of mobile agents. Wrapper agents provide an interface between mobile unit users and the AMDB platform. For that reason, Wrapper agents have the following functionalities. First, they should create the execution context for mobile agents. Thus, wrapper agents behave like a middleware between a mobile agent and a database participating of an MDBC. Second, wrapper agents provide a common data representation of the stored local data at mobile computers. As already said, XML will be used as a common data representation. Additionally, a wrapper agent running on a given mobile unit is also responsible for temporarily transferring one or more services to another mobile computer of the community to improve performance or whenever a critical situation occurs (for example, temporary service unavailability such as insufficient local memory for processing a query) at its mobile computer. In this case, any mobile agent visiting the mobile unit of the wrapper agent should be redirected to the unit where the service is being provided.

Agents of the mobile-agent class are responsible for implementing the logical mobility property in the AMDB architecture. Consequently, they should be able to transport themselves from one MDBC member to another. In the proposed architecture, the basic feature of mobile agents is to carry database access code and access results, while migrating between MDBC members. There are three types of mobile agents: Runner, Loader and Broker agents. Runner agents are directly responsible for performing tasks required by mobile unit users in remote mobile databases belonging to an MDBC. Such tasks can represent data queries, data updates or schema evolution of the mobile databases. Loader agents have the functionality of carrying a database query result back to the mobile unit that has required the query. Broker agents should gather schemas of mobile databases of an MDBC, when a mobile unit joins to an

MDBC. Moreover, a broker agent can define a mobile unit as temporary storage for storing partial query results when it is necessary.

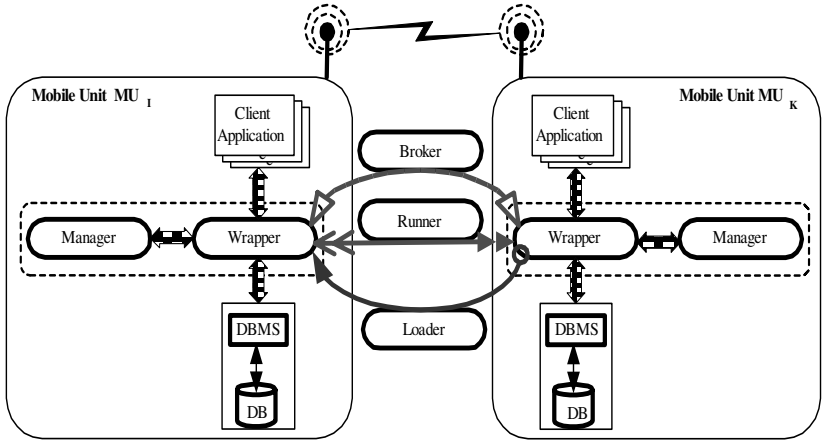

Fig. 1. Abstract Model of the AMDB Architecture.

Figure 1 depicts an abstract model of the AMDB architecture. In order to describe how the proposed architecture works, suppose that a mobile unit MU_K wants to form an MDBC. For that, it declares itself as initial coordinator of the new community to its wrapper agent. After that, the wrapper agent of MU_K sends information about the new community to all computers inside the area covered by the wireless communication network in which MU_K is connected. When news members begin to join to the new MDBC, the central coordinator role is not necessary anymore. This coordinator function will be performed local and collaboratively by the wrapper agents at each mobile unit which is member of the community.

Now, suppose that a mobile unit MU_I wants to join to the MDBC, which was initially created by MU_K. First of all, it is necessary an explicit declaration of MU_I in order to see the schemas of all databases which are member of the community. This functionality is executed by the local wrapper agent, which, in turn, creates a broker (mobile) agent and gives to it the task of querying local database schema at all mobile units, which are hosts of database systems. The broker agent will roam unit by unit and will collect the local schemas. Local database schemas will be provided by local wrapper agents. Recall that local schemas visible to the members of an MDBC are represented in XML Schema [23]. When the broker agent of MU_I returns to MU_I, it brings the schemas of the participating databases and passes them away to wrapper agent of MU_I. With the database schemas of each database of the MDBC, users or application programs at MU_I are able to access data stored at those databases through an interface provided by local wrapper agents. Therefore, users or application programs submit queries to the local wrapper agent. Such queries should be specified in an extension to the XQuery, called MXQuery [25]. The key feature of the MXQuery language is to provide mechanisms which support the capability to jointly manipulate data in heterogeneous data sources. The idea is to use the MXQuery as a multidatabase language [26] in an MDBC, making possible the integration of heterogeneous and distributed data sources. It is important to note that an MXQuery algebra has been

defined based on the XML formal semantics algebraic operators (formerly XML query algebra). A global query denotes a query, which should be executed over several databases of an MDBC. When the wrapper agent of MU_I receives a global query, it creates a runner (mobile) agent at MU_L and passes to the runner agent the MXQuery expression corresponding to the submitted query. After that, the runner agent begins to process the query. The query execution plan should be generated considering optimization opportunities and the hosts (MDBC's participants), where the query will be actually executed. It is important to observe that an MDBC is characterized as a distributed environment. Therefore, most queries in such an environment are distributed. In section 5, query processing performed by runner agents will be detailed.

The scenario described above is illustrated in Figure 1. The wrapper agent of MU_I creates a runner agent, which has to migrate to MU_K, considering that MU_K encompasses the target DBMS for processing the query submitted initially at MU_I. When the runner agent arrives at MU_K, it sends the MXQuery expression to the local wrapper agent. The wrapper agent of MU_K, in turn, maps the MXQuery expression to the native query language of the local database and submits it for execution as a common user. When the result is available, the local wrapper agent passes it back to runner agent. If the result is too huge and the runner agent has still to migrate to others units, the runner agent can create a loader agent and transfers to it the result of the query execution on MU_K. The loader agent begins a trip back to the runner-agent home unit. It is important to note that once the loader agent creation has finished, the runner agent can resume migrating to other units. Asynchronously, the loader agent takes the partial query result data back to MU_I. When the runner agent returns home, it communicates to the wrapper agent of MU_I that its task has finished. The wrapper agent unit shows the result to the user or sends the results to application program. Next agent functionalities will be described and analyzed.

4 Processing Queries in an MDBC

In section 4, we have seen that the wrapper agent is responsible for providing local database schema to all MDBC members. In existing database systems, such schemas consist of metadata and statistical data about the database such as, for example, tables' cardinality, tables' blocking factor and the height of index structures [20]. Statistical data are used for optimizing query processing. Hence, if a mobile unit wishes to participate in an MDBC, it has to send its broker agent to each mobile unit of the community providing its local schema and some statistical data. In the meanwhile, the broker agent can gather the local schemas of the visited units. Once the broker agent has returned home, the mobile unit has the necessary information for submitting queries over participant databases.

As already mentioned, users or application programs submit queries in the AMDB environment in MXQuery format. Such queries can be categorized in three classes: local, global (distributed) and remote queries. A local query only involves operations over objects of the local database. In this case, the local wrapper agent translates the MXQuery expression into the native query language of the local DMBS. After that, the wrapper agent submits the query to the local DBMS using the DBMS common user interface.

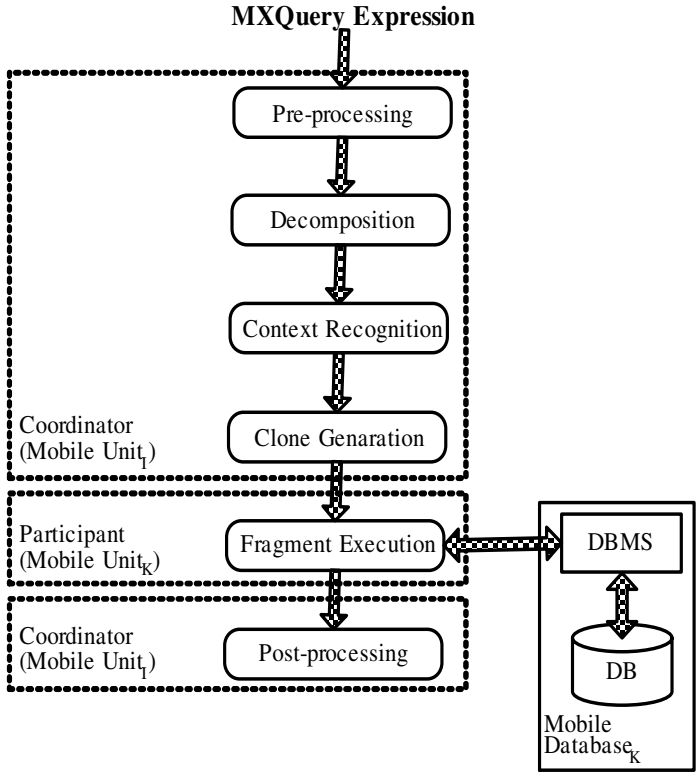

Fig. 2. Query engine architecture.

In turn, a global query may involve operations over several mobile databases of an MDBC. When a mobile computer user (or an application) submits a global query to the local wrapper agent, that agent, detecting a distributed query, creates a runner agent and delegates to it the functionality of executing the global query. It is important to observe that the runner agent receives the global query in MXQuery format. The runner agent migrates to mobile units in which the query should be executed.

Finally, a remote query encompasses operations which should be executed in a remote database belonging to an MDBC. In other words, a query submitted in MU_I should be executed over the databases of a mobile unit MU_K. Of course, those mobile units should belong to the same MDBC. A remote query is processed as follows. A user submits a remote query to the local wrapper agent, which, in turn, creates a runner agent and delegates to it the functionality of executing the remote query. For that purpose, the runner agent should migrate to the remote mobile unit in which the query should be executed.

Figure 2 depicts an abstract model of the query engine for processing global (distributed) and remote queries. According to the proposed engine, a query is processed by six distinct phases: Pre-Processing, Decomposition, Context Recognition, Clone Generation, Fragment Execution and Post-Processing phases. We denote Coordinator

the origin unit that has submitted the query. The functionalities of the coordinator are executed by different agents, such as runner, loader and wrapper agents. A Participant represents a DBS, which is responsible for executing locally operations of a global or remote query.

Observe that the proposed query engine is fully distributed, since query processing activities are executed by different components (or agents) in different sites. It is important to note that the query optimization process is carried out by using metadata about local database schemas, the number of hosts in which the query should be executed and available statistics information about data at each local.

Next, we describe the phases which should be implemented by the proposed query engine.

The activities of the **Pre-processing** phase are executed by wrapper and runner agents. Initially, the query submitted in an MXQuery format is parsed by the wrapper agent. After that, the wrapper agent sends the parsed query to the runner agent. After receiving the query, the runner generates a query execution plan (QEP). The generated QEP is represented by an operator tree [2,8], whose nodes represent MXQuery algebraic operators or data sources. The root represents the final query result. An intermediate node indicates the application of the corresponding operator on the results generated by its children. The edges of a tree represent the data flow from bottom to top, i.e., from the leaves to the root of the operator tree. The nodes have also attributes like data locality, selectivity factor and name of data source.

After the Pre-processing phase, the query engine starts performing the **Decomposition** phase. They goal in the execution of this phase is to decompose a query into fragments. Of course, in the case of execution of a remote query only one fragment is produced. The criterion for decomposing a query is the locality principle. Hence, each fragment should be executed in a given database belonging to an MDBC. In fact, a query fragment represents a sub-tree of the global QEP generated by the Preprocessing module. After the Decomposition phase, the **Context Recognition** phase is processed.

Once a query has been divided into fragments, the query engine has to verify, for each fragment, whether or not a given local DBMS has resources to execute the fragment. This the main functionality of the Context Recognition phase. The context recognition phase is carried out based on information about not allowed operations for each mobile unit. That information may be provided by mobile units at moment when a mobile unit joins to an MDBC. If such information is provided, it means that local DBMS cannot perform the defined operation set. Otherwise, it means that local DBMS does not have any restriction about query operations.

If a given DBMS does not have support to execute any operation, the query engine itself is responsible for executing that operation. In fact, this should be performed by the runner agent and can be carried out either on the origin unit or on the mobile unit selected as temporary storage unit (see bellow). In this case, it is necessary a QEP reordering for bringing to high-level nodes (nodes close to the root) of the QEP the operations, which should be executed by the Runner agent.

Another functionality of the context recognition phase is to define a temporary storage unit for materializing intermediate query results. This should be done when the origin unit and other units referenced in the QEP have limited computational resources, such as main memory or disk space. In this case, such hosts do not have the necessary computational support nor to store partial result neither to execute high-

level operations specified in the QEP. The step of defining a temporary storage unit is performed as follows. First, the query engine verifies whether or not the origin unit has enough resources for executing the high-level operations. If not, it verifies the other units referenced in the QEP. If none of the units can be defined as temporary storage, then an error message is sent to the user (or application program) indicating the unavailability of resources for executing the query.

Once Context Recognition phase has been finished, the **Clone Generation** phase is started. In this phase, the runner agent clones itself (cloning is supported by several agents platforms). A clone for each query fragment (generated during the Decomposing phase) is created. Each clone is responsible for executing a given query fragment. After the clones have been created, they are dispatched (carrying a query fragment) to the mobile units in order to execute the query fragments.

The **Fragment Execution** phase is started when a runner agent (or its clones) migrates to the mobile units in which query fragments of a given query should be executed. On arriving at a given mobile unit, the runner agent or a clone submits to the local wrapper agent the query fragment in MXQuery format. The wrapper agent, in turn, translates the fragment into the native query language of the local DMBS. After that, it submits the query to the local DBMS and returns the results back to the runner or clone. Obviously, local DBMSs can further optimize the processing of the query fragment it is executing.

During the execution of the Fragment Execution phase local statistical information can be gathered by the runner agent (or clone), since information about operation, cardinality of tables or of selections, for example, can be inferred while the query fragment is being executed. When the runner or clone returns to the origin unit, it sends the collected statistical data to the wrapper agent of the origin unit (coordinator) for updating statistic information about of the mobile unit in which such information was obtained.

The last phase is the **Post-processing**. This phase is initiated when the clones begin to arrive at the mobile unit, on which they were created. During this phase, the clones give to the runner agent the results of the query fragments. The runner agent, in turn, will start the execution of operations that could not be executed by local DBMSs. In this case, the runner agent performs operations corresponding to the high-level nodes of the operator tree. Once the execution has finished, the runner agent joins the result of the query fragments. After that, the runner agent passes the final result to the wrapper agent. The wrapper agent, in turn, is responsible for projecting the final result to the user or application which has submitted the query. It is important to note, that statistic information is also gathered during the execution of this phase.

5 Mobile Transaction Processing

Mobile users interact with databases in an MDBC by invoking transactions. A transaction represents a sequence of operations on database objects. There are two types of transactions in an MDBC environment: local transactions and global transactions. A local transaction is submitted directly to a mobile database DB by an application running on the same host on which DB resides. A global transaction consists of database operations which should be executed on different hosts belonging to an MDBC.

Therefore, the notion of global transaction models distributed transactions which are composed by mobile sub-transactions. In our approach, the correctness criterion for the execution of concurrent transactions is serializability. We assume that each mobile database in an MDBC guarantees serializability by the strict version of the two-phase locking (2PL) protocol [1]. This is a quite reasonable assumption, since all existing database systems implement that protocol.

In the proposed architecture, the runner agents are responsible for managing the execution of global transactions. In order to process mobile transactions, an extension to the conventional centralized version of the 2PC protocol [1] is proposed. The idea is to consider the mobility property, when processing distributed transactions over several mobile databases. The proposed extension is called mobile 2PC protocol (M2PC).

Next, we describe how the M2PC protocol works. Consider that a database system DBS_I resides on a mobile unit MU_I, which belongs to an MDBC. Suppose that a user submits a global (distributed) transaction T to MU_I. In fact, the transaction T corresponds to the execution of a query, which involves operations over several databases of the MDBC. Thus, a runner agent R is created by the wrapper agent of MU_I (see section 4). The runner agent R is responsible for managing the execution of the transaction T. For that reason, R is said to be the coordinator of T. Before starting the execution of T, the agent R creates an agent whose functionality is to play the role of a backup for the M2PC coordinator process. This is necessary because the runner agent R (coordinator) may visit a mobile unit which can disconnect from the wireless communication infrastructure or exit the mobile database community. In this case, a backup process should assume the coordination of the extended 2PC protocol.

The backup for the M2PC coordinator process is created as follows. Before starting the execution of a global transaction, the runner agent creates a broker agent and dispatches it to the MDBC members. The broker should select one of the visited mobile units to be the host of the agent, which has functionality of behaving as a backup for the M2PC coordinator. When the broker agent returns home, it passes the network address of the selected unit to the runner agent. The runner agent creates a clone, passes to it the list of units in which the operations of the global transaction have to be executed, and dispatches the clone to the select unit. That clone has the functionality of being the backup of the M2PC coordinator process. It is important to observe that the creation of that backup for the coordinator process reduces the frequency of executing a termination protocol [1] for the M2PC protocol. In the AMDB architecture, the termination protocol has to be triggered, if and only if the M2PC coordinator and its backup are unreachable.

After creating the backup for the coordinator process, the runner agent creates n–1 clones. It is important to note that each clone has the network address of the mobile unit which is the host of the backup coordinator. The clones migrate to the hosts on operations of the global transaction should be executed. The runner agent migrates to a mobile unit as well. When the clones return home, they bring the votes corresponding to the second step of the conventional 2PC protocol. They send the votes to the runner agent (coordinator for the execution of T), if it has already come back.

When a 'NO' vote is detected, the runner agent sends a message to the backup coordinator with the decision for aborting transaction. After that, the runner agent can send a message with the final decision for aborting transaction to each non-returned clone or can dispatch once more the clones to each unit already visited with the final decision for aborting transaction. If all of votes were 'YES', the runner agent sends a

message to the backup coordinator with the final decision for committing the transaction. The clones are dispatched again to the MDBC units with the final decision for committing the transaction.

If the runner agent is unreachable, the backup coordinator assumes the coordination process. In this case the new coordinator decides to abort the transaction. It sends a message to the clones and dispatches them to the transaction-involved units with the decision for aborting transaction.

6 Conclusions

In this paper, we define the concept of dynamically configurable database communities in mobile computing environments. According to that concept, a dynamic collection of autonomous mobile databases, interconnected through a wireless communication infrastructure, can be opportunistically formed.

In order to provide a platform to enables such communities to be formed opportunistically over mobile database hosts in ad hoc configurable environments, an architecture is described and analyzed. The proposed architecture, denoted AMDB (Accessing Mobile Databases), supports physical mobility of hosts and logical mobility of database queries (or transactions) and their results across mobile hosts. Moreover, the proposed architecture has important additional properties for coping with database mobility, such as:

i. it does not require changes to the core of the underlying database systems;
ii. it supports opportunistic creation of database communities over a varying number of mobile and autonomous database hosts.

For query processing in mobile database communities, the proposed architecture implements a query optimizer that adapts the query plan to the execution scenario of a given query. With respect to transaction processing, the 2PC protocol was extended to take into account the mobility property. However, we are investigating other transaction models less restrictive than serializability-based model [1]. Particularly, we are interested on the semantic serializability model proposed in [3] and [4].

A prototype of the AMDB architecture is currently being developed based on IBM AGLETS platform, a Java-based platform which supports logical mobility.

References

1. Bernstein P., Hadzilacos V., and Goodman N.: Concurrency Control and Recovery in Database Systems, Addison-Wesley, 1987.
2. Bouganim L., Fabret F., Mohan C. and Valduriez P.: A Dynamic Query Processing Architecture for Data Integration Systems. IEEE Data Engineering Bulletin. Vol. 23 No. 2. June-2000.
3. Brayner A., Härder T. and Ritter N.: Semantic Serializability: A Correctness Criterion for Processing Transactions in Advanced Database Applications. DATA & KNOWLEDGE ENGINEERING, 31, 1999.
4. Brayner A. and Härder T.: Global Semantic Serializability: An Approach to Increase Concurrency in Multidatabase Systems. Cooperative Information Systems - Lecture Notes in Computer Science. 301–315. Springer-Verlag, 2001.

5. Dunham M. H. and Kumar V.: Impact of Mobility on Transaction Management. In Proceedings of the International Workshop on Data Engineering for Wireless and Mobile Access, August 1999. pp. 14–21.
6. Emerich W., Mascolo C. and Finkelstein A.: Implementing Incremental Code Migration with XML. Proceedings of the 22nd International Conference on Software Engineering. Limerick, Ireland. 2000. pp. 397–406.
7. Holliday J., Agrawal D. and A. Abbadi E.: Planned Disconnections for Mobile Databases. In Proceedings of the 11th IEEE International Workshop on Database and Expert Systems, 2001.
8. Ives Z., Florescu D., Friedman M., Levy A and Weld D. S.: An Adaptative Query Execution System for Data Integration. In Proceedings of ACM SIGMOD International Conference on Management of Data, Philadelphia, USA. 1999. pp. 299–310.
9. Kemper A. and Wiesner C.: HyperQueries: Dynamic Distributed Query Processing on the Internet. In Proceedings of 27TH International Conference on Very Large Databases, Roma, Italy. 2001. pp-551–560.
10. Lange D. B. and Oshima M.: Programming and Deploying Java Mobile Agents with Aglets. Addison-Wesley. Massachusets, USA. 1998.
11. Jing J., Helal A. and Elmagarmid A.: Client-Server Computing in Mobile Environments, ACM Computing Surveys, Vol. 31, No. 2, pp. 117–157. June 1999.
12. Macker J. P. and Corson M. S.: Mobile Ad Hoc Networks and the IETF. Internet Engineering Task Force, MANET Working Group. Online at http://www.ietf.org/html.charter/manet-charter.html.
13. Manolescu I., Florescu D., Kossman D.: Answering XML Queries over Heterogeneous Data Sources. In Proceedings of 27TH International Conference on Very Large Databases, Roma, Italy. 2001. pp. 241–250.
14. Mendonça N. C., Brayner A. and Monteiro J. M.: Mobile Database Communities: An approach for Sharing Autonomous Mobile Database in Ad Hoc Networks. Submitted for publication. 2002.
15. Murphy A. L., Pico G. P. and Roman G.-C.: Lime: A Middleware for Physical and Logical Mobility. Proceedings of the 21st International Conference on Distributed Computing Systems. April 2001. pp. 524–533.
16. Özsu M. T. and Valduriez P.: Principles of Distributed Database Systems. 2nd Edition Prentice Hall, 1999.
17. Patel J. M.: Query Processing in Mobile Environments. NFS Wokshop on Context Aware Mobile Database Management (CAMM). January 24-25, 2002, Providence, Rhode Island, USA. Online at http://www.sice.umkc.edu/nsfmobile/wshop.html/JigneshMPatel.pdf.
18. Phatak S. H., and Badrinath B. R.: An Architecture for Mobile Databases. Technical Report #DCS-TR-351. Rutgers University, Department of Computer Science. Online at ftp://www.cs.rutgers.edu/pub/technical-reports/dcs-tr-351.ps.Z.
19. Roman G.-C., Pico G. P. and Murphy A. L.: Software Engineering for Mobility: A Roadmap. In A. C. W. Finkelstein, editor: Future of Software Engineering. ACM Press, 2000.
20. Silberschatz A., Korth H. F. and Sudarshan S.: Database System Concepts. 3rd Edition McGraw Hill, 1998. ISBN 0070310866.
21. Singhal M.: Techniques for Building Large Relational Databases on Mobile Computing Systems. NFS Wokshop on Context Aware Mobile Database Management (CAMM). January 24–25, 2002. Providence, Rhode Island, USA. Online at http://www.sice.umkc.edu/nsfmobile/wshop.html/MukeshSinghal.pdf.
22. Vlach R., Lána J., Marek J. and Navarra D.: MDBAS – A Prototype of a Multidatabase Management System Based on Mobile Agents. Proceedings of the 27th Annual Conference on Current Trends in Theory and Practice of Informatics (SOFSEM), Springer-Verlag, Nov. 25 – Dec. 2, 2000. Milovy, Czech Republic.
23. XML Schema. Online at http://www.w3.org/XML/Schema.

24. XQuery 1.0 Formal Semantics. Online at http://www.w3.org/TR/query-semantics/.
25. Soares M., Brayner A.: MXQUERY: An XQuery-like Multidatabase Language. Submitted for publication. 2002.
26. Grant J., Litwin W., Roussopoulos N., Sellis T.: Query Languages for Relational Multidatabases. In Proc. of 19th VLDB Conference. 1993.