A Meta Protocol for Adaptable Mobile Replicated Databases[1]

Udo Fritzke Jr.[1], Luiz Alberto F. Gomes[1], Denis L. Silva[1], Daniel M. Morais[1]

[1]Pontifícia Universidade Católica de Minas Gerais
Curso de Ciência da Computação
Poços de Caldas, MG - Brazil

{udo,luizgomes}@pucpcaldas.br, denislimamg@yahoo.com.br
dmerlimorais@gmail.com

Abstract. *This paper presents a meta protocol that allows the replacement of replication control protocols in mobile replicated databases. The meta protocol is motivated by the adaptability requirements of mobile database systems but can also be used on replicated databases with fixed nodes. The paper defines three properties that meta protocol executions have to enforce and specifies a protocol that satisfies these properties. The protocol is based on transactions and atomic broadcast. Finally, we outline a dynamic adaptable architecture that includes the meta protocol and that is based of an aspect oriented framework and on a group communication system.*

1. Introduction

Data replication has been widely used in order to provide database systems with high availability and better performance in a distributed environment. In mobile database systems, data replication allows a mobile node to access local data while it is disconnected from a network. During the execution of a replicated system, the access of a replicated object is controlled by a particular replication control protocol. Such a protocol collaborates with concurrency control protocols so that data accesses are synchronized offering a one-copy view of the system [Bernstein *et al*., 1987]. There exists a large variety of replication control protocols [Wiesmann *et al.*, 2000] that have been conceived upon different assumptions on data consistency (or transaction isolation) needs, communication primitives, node disconnection modes (failures or planned disconnection), replication models and data synchronization methods.

The adaptation of replication techniques to particular application needs has also been subject to study (*e.g.* [Drapeau *et al.*, 2002]) and is motivated both by the miscellanea of replication protocols and also by environment changes to which an application can be subjected. These include modifications of requirements as confidentiality of data, data placement, data administration policies, storage capacity, and data access latency, among others. As mentioned in [Drapeau *et al.*, 2002], we understand that there is no practical replication control protocol that fulfills all adaptation needs, even if it was conceived to be parameterized. Moreover, we believe that runtime adaptation is important to some applications and accordingly, it should be part of database replication frameworks.

As part of our motivation, we borough the application scenarios presented in [Holliday *et al*., 2000]. Holliday *et al.* show three practical replication schemes for mobile replicated databases. In the first one, *basic sign-off* mode, a mobile node

[1] This work has been developed with the support of FAPEMIG (Fundação de Amparo à Pesquisa do Estado de Minas Gerais), CEX-818/05.

disconnects from the network keeping local copies of read-only data. When it reconnects it synchronizes the state with the remaining nodes that have possibly updated the database. In a second scheme, *check-out* mode, a mobile node can travel with updatable data, usually a portion of the database, while other nodes are not allowed to access the data that was checked-out by the mobile node. When the node reconnects it updates the remaining database copies with the new state of the data. These two modes provide strong consistency (one copy-serializability). A third method, *relaxed check-out* mode, relaxes the previous one by allowing connected nodes to "browse" the data that was checked-out by the mobile node. This ensures a lower consistency level as long as nodes can browse old data. As a practical scenario, one could imagine that during an usual activity period, nomadic users access portions of data in relaxed check-out mode. When all of nodes reconnect to an enterprise network, the administrator of the database claims to himself the data access starting a *basic sign-off* mode, so that he or she is allowed to execute data management activities, as for example, users inclusion and data schemata alteration. Therefore, replication control protocol replacement would be an important requirement for such a system.

We treat in this paper the dynamic replacement of replication control protocols, so that such an adaptable scenario could be implemented. We consider that a protocol replacement can be requested by any node of the system, and these requests can occur concurrently. In this paper, while the term "protocol" denotes distributed replication control algorithms, the term "meta protocol" has been adopted to denote protocols that maintain the consistency of the replication control protocols despite concurrent replacement requests.

The rest of the paper is structured as follows. Section 2 presents definitions and models of mobile databases, concurrency and replication control, transactions and group communication. Section 3 presents three safety properties of the meta protocol. Section 4 details the meta protocol, and section 5 briefly describes a dynamic architecture that is being implemented. Section 6 concludes the paper.

2. Models and Definitions

2.1 Mobile Database and Failures

We consider that the system is composed of nodes that may replicate objects (or data items). A node, noted as N_i, can disconnect from a network and possibly access local data during disconnection. Objects are accessed by read and write operations executed on behalf of a transaction, noted as t (or t_i to avoid ambiguity), following the execution model presented in [Bernstein *et al*., 1987]. Two transactions are concurrent if one of them begun its execution before the termination (with *commit* or *abort*) of the other. Two operations conflict if they belong to different transactions, access the same object and one of them is a write operation. Two concurrent transactions are said to be conflicting if both access some object (replicated or not) with conflicting operations.

During a disconnection period of a node, data accesses are performed on local copies, and no synchronization between nodes is possible. After reconnection a node can re-synchronize the state of local copies with respect to other nodes' copies. Disconnections and re-connections are planned by the user or due to a failure. Objects residing in a node are accessed through a Database Management System (DBMS).

We assume that nodes only fail by prematurely stopping their execution (*crash* failure model) [Hadizlacos and Toueg, 1993]. A node (or process) is said "correct" if it never crashes. We use "node" instead of "process" for the sake of notation simplicity.

2.2 Concurrency and Replication Control Protocols

Data access isolation is guaranteed by concurrency control protocols. These protocols handle operations that access shared data (*writes* and *reads*) so that execution histories respect the required isolation level, and therefore, ensure some data consistency criterion. The literature presents several concurrency control protocols, some of them for strong consistency levels (as two-phase locking [Bernstein *et al.*, 1987]) and other for more relaxed consistency. In the latter case, the protocols are designed to avoid deadlocks (*e.g.* [Gray *et al*., 1996]) and to minimize data access latency in mobile systems (*e.g.* [Holliday *et al*., 2000][Shapiro *et al.*, 2000] [Fritzke *et al.*, 2004]). The choice of the algorithm is strongly related to the integrity constraints of application's data.

The coherency of object copies is ensured by replication control protocols. These protocols aim to make an execution composed by operations that access replicated data appear as an execution on a non replicated system. Traditional replication control protocols use to synchronize data accesses during transaction executions (*e. g.* [El Abbadi *et al.*, 1986]). However, in mobile settings transactions cannot be synchronized when a node is disconnected from the network. Therefore, different protocols that allow local execution of data accesses and further dissemination of the updates after reconnections have been proposed (*e.g.* [Gray *et al*., 1996], [Holliday *et al*., 2000] and [Fritzke *et al.*, 2004]). These two synchronization methods are defined in [Gray *et al*., 1996] as *eager* and *lazy* replication, respectively.

Concurrency and replication control protocols normally cooperate in order to achieve the desired isolation and coherency levels (*e.g*., one-copy serializability). For the sake of notation simplicity, we will refer to these protocols simply as *replication control protocols* in the rest of the paper.

We consider that any transaction operation submitted to a DBMS is handled first by a replication control protocol. The replication control protocol will then be allowed to schedule transaction executions as well as disseminate transaction updates according to its concurrency and replication control policies.

2.3 Distributed Transaction Execution

An application at node N_i can possibly connect to a local DBMS and execute local reads and writes while it is isolated from a network or from some other node of the distributed system. When N_i reconnects to some node it has to synchronize its database with respect to the database state of other nodes. Therefore, we consider that a transaction on a replicated environment passes two distinct and sequential execution phases:

- *Local phase*. Access operations are executed on the local DBMS. Most replication control protocols allow only read operations, but there are other protocols that also allow data modifications. In mobile database applications, frequently, transaction commit is constrained to local accesses and so they are instead executed in the local phase (lazy replication). The local phase of a transaction is initiated by the first operation submitted to the DBMS.
- *Global phase*. The goal of this execution phase is twofold: (1) to disseminate the effects of writes on local data to remote copies and/or from remote copies to the

local ones, and (2) to perform an atomic distributed commit in order to provide a transactional semantics of data accesses (eager replication). So, a node can possibly participate in the global phase of remotely initiated transaction in order to update local and remote copies of objects and to execute a distributed atomic commit to terminate the remote transaction.

The number of nodes involved in the global phase may vary from one DBMS server up to all nodes of the distributed system. The former is normally the case of common client-server applications, while the latter corresponds to fully replicated databases. Our meta protocol assumes that all database objects are fully replicated.

We consider that the global phase can occur progressively, so that all nodes that replicate an object modified by N_i eventually get updated with N_i's modifications, and vice-versa. Therefore, we define here a *global phase termination* event that represents the dissemination of writes followed by the distributed atomic commit of a transaction at all nodes. This execution model is not completely general as is does not include synchronization that could occur during earlier accesses to distributed data by a transaction, what happens for example in writes on a ROWA (Read-One Write-All) or quorum basis. In most cases, however, one could postpone writes to the global phase, after all reads are executed on local data. Moreover, synchronization at the transaction termination phase is particularly interesting for mobile databases, where nodes are frequently isolated from communication networks. In some mobile applications, the set of transactions executed in a mobile node while it is disconnected, can in fact be perceived by other nodes as only one transaction execution whose global phase happens when the node reconnects to the system.

This abstract *global phase termination* event allows the decoupling of the actual implementation of replication control protocols from the meta protocol we will propose in section 4. This is important as there are numerous protocols that could be handled by the meta protocol. We will discuss in section 4.3 how a replication control protocol can launch such an event.

2.4 Atomic Broadcast

We use in our protocol an *atomic broadcast* primitive [Hadizlacos and Toueg, 1993]. A node that wants to broadcast a message *m* to all nodes of the system executes an `A_Broadcast(m)`. The delivery of *m* to a node is *reliable*. Informally, this means that when either a correct node broadcasts a message *m* or some node delivers a non spurious message *m*, then all correct nodes eventually deliver *m*. Moreover, message deliveries respect a *total order* property: if two nodes N_1 and N_2 deliver two messages *m* and *m'*, then N_1 delivers *m* before *m'* if and only if N_2 delivers *m* before *m'*.

3. Properties for Protocol Replacement

In this section we present properties that our meta protocol has to enforce. In a similar way as presented in [Yang and Li, 2004], we consider that during its execution, a replicated system can pass through three sequential execution phases: *protocol attachment, protocol execution* and *protocol detachment*. The first phase consists of reconfiguration steps that are performed by the meta protocol to install a replication control protocol in all nodes of the distributed system. The second phase corresponds to the application execution on top of an installed replication control protocol, during which accesses to replicated objects are synchronized according to the rules determined by the replication control. The third phase consists of the steps that are performed by the meta

protocol to remove a previously installed replication control protocol from all nodes of the distributed system. Once a replication control protocol is first attached, a subsequent attachment of a new protocol is only allowed after the previous protocol is detached.

With respect to mobility of nodes, we consider that the dynamic replacement only takes place when all nodes are connected to a network and allowed to communicate among themselves.

Replication control protocol replacement is constrained by the execution state of transactions. A new protocol cannot be negligently attached to a node while transactions are active and under control of another protocol. If we consider a distributed execution in which several replication protocols are attached to nodes and run, we need to understand when these protocols can be attached, run and detached. In order to clarify this dynamic replacement, we present three safety properties that relate replication control protocols with transaction executions:

Object copy access consistency. At any time in system's execution, any copy of an object is accessed by a transaction under control of only one replication control protocol. In other words, each node with an object copy runs only one protocol at a given time.

Transaction control consistency. The execution of a transaction, including the local and the global phases, is controlled by the same replication control protocol, attached to all nodes that replicate objects accessed by the transaction. When a transaction execution satisfies this property, we say that the transaction is *protocol consistent*.

Object access consistency. Consider any two concurrent and conflicting transactions t_1 and t_2 that are protocol consistent, and let O be the set of objects on which t_1 and t_2 execute conflicting operations. During t_1 and t_2 executions (*i.e.* operations of the local phase plus the operations of the global phase), both are subject to the same replication control protocol. In other words, all nodes that hold copies of the objects in O have attached to themselves the same replication control protocol during the execution of t_1 and t_2.

The first property ensures that no two replication control protocols can mediate the execution of a transaction accessing an object copy during application execution. The treatment of transaction operations at some node is done deterministically by exactly one protocol. This property is important in the sense that it avoids the attachment and execution of a protocol in a node, without the detachment of the previous one. In a practical implementation, for example using a dynamic weaving mechanism in which protocols are implemented as advices intercepting the execution of the system, no two advices responding to distinct protocols should be deployed. Section 5 will present an architecture based on an aspect oriented framework. Figure 1(a), that shows three nodes replicating some object and that changed replication control from "protocol 1" to "protocol 2", illustrates this reasoning. Node 1 appears as it had installed simultaneously the two protocols, creating a protocol conflict situation and thus violating the *object copy access consistency* property.

The second property ensures that a transaction execution that is started under some replication control protocol is constrained by that protocol until it is completely executed. Figure 1(b) shows a transaction execution in which the local phase is executed under some (replication control) "protocol 1". The global phase is also initiated under the same replication control protocol, but it terminates in node N_3 under control of "protocol 2". Supposing that protocol 1 and 2 differ in the isolation level they implement, it is clear

that the overall execution of the transaction could not be consistent by none of the isolation levels. This cannot happen to a *protocol consistent* transaction.

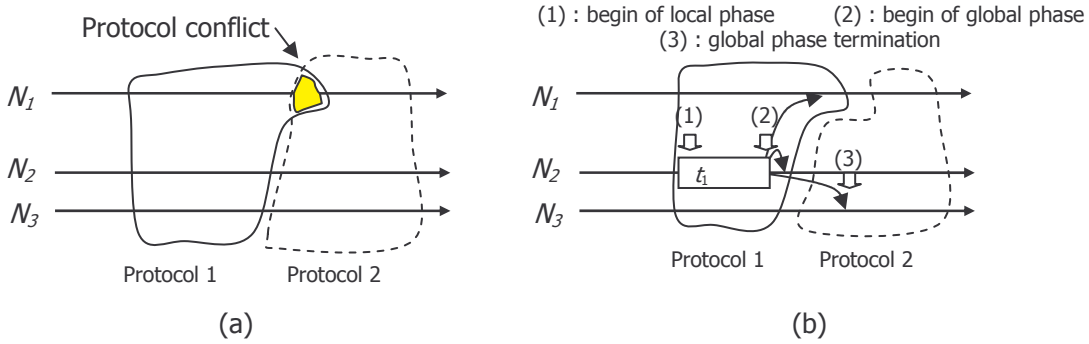

Figure 1 – (a) Violation of the *object copy access* consistency property, and (b) violation of the *transaction control consistency*.

The third property extends the transaction control property to concurrent and conflicting transactions. Figure 2 illustrates two execution scenarios that exemplify the importance of this property. It shows two transactions, t_1 that reads and writes into replicated objects x and y, and t_2 that writes into y. All operations of t_1 are controlled at nodes N_1, N_2 and N_3 by "protocol 1", thus ensuring that the transaction is protocol consistent. Moreover, all operations of transaction t_2 are controlled by a "protocol 2", also ensuring transaction control consistency with respect to t_2. That scenario could clearly lead to data integrity problems when for example the concurrency is controlled by distinct protocols, say using an optimistic approach for t_2, and a pessimistic one for t_1. When updates of t_2 reach node N_1 they can be considered correct by a certification phase, as the optimistic protocol does not care about locks hold on behalf of t_1, an then they are committed into the local DBMS. On the other hand, the updates of t_1 will also ignore the certification of t_2, and proceed with t_1's execution that overwrites the copy of y at all nodes. Such a phenomenon is avoided if object consistency is implemented by the system and this because both transactions would be submitted to the same replication control protocol. We remark that if both transactions did not conflict, they could be executed under different replication control protocols. For example, if t_1 and t_2 were both read-only transactions, they could be executed no matter what protocol was controlling each of them. This is due to the fact that in this case, their executions are "naturally" isolated one from the other and so they are inoffensive to data consistency and coherency.

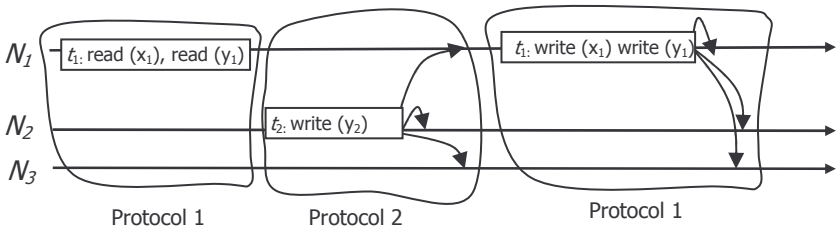

Figure 2 – *Object consistency* violation scenario due both to a conflict of transactions t_1 and t_2 on the access of objects *x* and *y*, and to a different replication control protocol for t_1 and t_2.

In the framework presented by Yang and Li [Yang and Li, 2004], protocols are attached to objects, and not to nodes, as we propose in our work. The fact that transactions that access different sets of objects could be executed under distinct replication control shows that Yang and Li's approach could be more interesting. Attaching protocols to nodes (instead of objects) could hidden some flexibility and concurrency gains when two protocols are allowed to coexist in a node, each responding to some disjoint collection of objects. However, from a practical point-of-view this leads to some problems. In such a scheme, a protocol consistent transaction would have to pre-declare the set of objects it intended to access so that the system could associate the same protocol to all accessed objects. This would constraint the execution of transactions that are dynamically defined by the user. Alternatively, the system could change the protocols attached to the objects as the transaction executes, what certainly would restrict the throughput of the transactional system. We note that Yang and Li's framework was designed for groupware applications that do not use to access shared objects by the means of transactions (instead each operation is treated as an atomic execution unit). So, the attachment of protocols to objects does not present the above mentioned drawbacks in such systems.

4. The Meta Protocol

4.1 Protocol Definition

In this section we present a meta-protocol that allows the dynamic replacement of replication control protocols and that implements the properties defined in the previous section. Initially, we consider that during its execution in some node of the system, a replication control protocol can be found in one of the three following states:

1. *Detached*: the protocol does not handle any operation on behalf of transactions submitted to the system.
2. *Attached and pending*: the protocol was requested by some node to become active but the meta protocol has not activated it yet in order to ensure the safety properties.
3. *Attached and active*: the protocol previously attached to a node becomes active and gets run to handle operations of transactions that are submitted to a DBMS.

Each node of the system maintains a queue, noted as Q, of structures recording relevant information about replication control protocols. Such a structure, denoted as *proto*, has the identification of a protocol, the *proto_id* field, and its state, the *state* field. A node N_i also manages a set with the identification of active transactions, denoted Act_i. A transaction is considered to be active if it has initiated its local phase but not yet terminated its global phase. At each node N_i, Act_i identifies local transactions and remote transactions for which the global phase has include N_i but has not been globally terminated so far.

In the figures that follow, the instructions in a scope defined by `begin` and `end` keywords are executed atomically, except during a blocking `wait` operation.

Accordingly, Figure 3 defines the rules that maintain the Act_i set at a node N_i. For any transaction that is initiated, its identification is atomically broadcast to all nodes. This allows the nodes to be aware of a transaction begin. The termination event handled in Figure 3 is launched by the active replication control protocol, as discussed in section 4.3.

```
1     when an operation of a transaction t is submitted to the DBMS
2       begin  A_broadcast(<t>) end /* broadcast t's identification */
3     when N_i detects the termination of t
4       begin  Act_i ← Act_i − t end
5     when N_i delivers <t> /* message delivery event */
        begin
6         Act_i ← Act_i ∪ t
7         Submit t to the replication control protocol
        end
```

Figure 3 – Rules for the management of the Act_i set at a node N_i.

Based on the state of active transactions we can coordinate the protocol reconfiguration at a node of the system. For that purpose we define two procedures that an application can call to request a protocol attachment or detachment, namely `Attach(`*proto_id*`)` and `Dettach(`*proto_id*`)`, where *proto_id* corresponds to the identification of the protocol to be attached or detached, respectively. Figure 4 defines the attachment procedure and the treatment of the protocol replacement requests that are delivered to a node.

```
      Attach(proto_id) /* attachment procedure */
1       begin A_broadcast(<attach, proto_id>) end
2     when a <attach, proto_id> message is delivered to N_i:
        begin
3         proto.proto_id ← proto_id
4         proto.state ← pending
5         put proto in queue Q
        end
```

Figure 4 – Replication control protocol attachment request at node Ni.

The procedure of Figure 4 is straightforward. A protocol attachment request is started by an atomic broadcast sent to all nodes (line 1). When a request is delivered to a node, it results in a new entry in the queue with information about the new protocol to be attached (lines 3-5). The state of the protocol is initially *pending* (line 4), and it becomes *active* after the detachment of previously attached protocols (Figure 5). Figure 5 defines the detachment procedure and the treatment of protocol detachment request.

```
      Dettach(proto_id) /* Dettachment procedure */
1       begin A_broadcast(<dettach, proto_id>) end
2     when a <dettach, proto_id> message is delivered to N_i:
        begin
          /* ensure that no transaction is under control of the protocol to be removed*/
3         wait while Act_i ≠ ∅
4         Stop the execution of proto_id
5         Let proto be the structure in Q that describes proto_id:
6           Remove proto from Q
7         Let proto be the oldest structure in Q:
8           proto.state ← active
9           run the proto_id protocol;
        end
```

Figure 5 – Replication control protocol detachment request at node N_i.

The procedure of Figure 5 shows that protocol detachment request is also redirected to all nodes by an atomic broadcast. After the delivery by a correct node, the

request results in the deletion of the protocol from the Q queue and in the activation of a new protocol following a FIFO policy (lines 4-9).

The wait condition of line 3 ensures that all active transactions run over the same replication control protocol, and this includes conflicting and non conflicting transactions. However, if there are concurrent transactions that do not conflict with each other, then there is no need to wait for the complete termination of these transactions before a protocol is removed and a new protocol is attached and activated. Accordingly, a node could instead attach a replication control protocol to each transaction and let these protocols run independently so far no conflicting transaction appears in the overall execution. However, in order to detect such a concurrent execution it would be necessary to ensure that no object is accessed with conflicting operations in a system wide base, and this can only happen after the global phase of transaction executions. Thus, our protocol waits for the global phase termination of all active transactions.

4.2 Discussion on the Meta Protocol Correctness

We informally discuss here the protocol safety in relation to the three properties defined in section 3. We assume that initially the same replication control protocol is installed at all nodes and no transaction has been started so far.

Object copy access consistency. It is easy to note that an attachment request results only in a new entry into the Q queue (Figure 4, lines 1 and 3-5) of a node. The requested protocol is only activated as the result of a detachment request (Figure 5), in which the protocol is activated (Figure 5, lines 8-9) only after the previous one is removed (Figure 5, line 6). Also, this is valid for all nodes that deliver an attach request message. Consequently *object copy access consistency* is preserved by the meta protocol.

Transaction control consistency. Suppose that a transaction t is started at a node N_i and that a detachment request is started by a node N_k. We have to consider two situations, (1) $k = i$ and (2) $k \neq i$.

In the first case, if the delivery of t (Figure 3) occurs before the delivery of the detachment request (Figure 5), then t is first included in the Act_i set and submitted to a replication control protocol (Figure 3, lines 6-7). The wait condition (Figure 5, line 3) allows a protocol withdraw (Figure 5, line 6) and a subsequent activation of a new one (Figure 5, lines 8-9) only after all active transactions are terminated, *i.e.* the transactions are removed from *Act* (Figure 3, line 4). Therefore, until t does not terminate no replication control protocol is removed at N_i. Otherwise, if the delivery of t (Figure 3) occurs after the delivery of the detachment request (Figure 5), then the requested protocol will be activated (as $Act_i = \varnothing$) and t will run under the control of the requested protocol, as lines 4-9 of Figure 5 are executed atomically. In both cases, t will execute entirely over the same replication control protocol.

In the second case, even with t and the protocol detachment request being issued by distinct nodes, the total order property of atomic broadcast will ensure that all nodes will treat the delivery of t's identification (and its inclusion into Act_i) and the protocol detachment request in the same order. Accordingly, all nodes will behave as N_i, and the points discussed above are also valid for this case. A more detailed proof can be obtained by extending the set of transactions and detachment requests and by taking the principles above as base steps. So, *transaction control consistency* is ensured by the meta protocol.

Object access consistency. Consider two conflicting transactions, say t_1 and t_2, that are executed concurrently. We will show that they are executed under control of the

same replication control protocol. We will take into account two situations: (1) t_1 and t_2 were started at the same node, and (2) t_1 and t_2 were started at different nodes.

In the first case, we show by contradiction that if one transaction is controlled by some replication control protocol, the other one could not be controlled by other protocol if they are conflicting and concurrent. Both transactions identifications are atomically broadcast when they are submitted, and their executions by a replication control protocol only occur after they are included in the set Act_i (Figure 3, lines 6-7). Suppose that t_1's identification delivery to some node, occurs first (Figure 3, line 5) and so, it is the only one that is in Act_i. Now consider that before t_2's identification delivery occurs, a protocol detachment request is treated by the protocol, resulting in a protocol replacement. This would only be possible if the termination event of t_1 was detected be the meta protocol (Figure 3, line 3) and that transactions was excluded from Act_i (Figure 3, line 4) so that the set is empty. In such a situation, t_1 was terminated and, as it was completely executed before t_2 they were not actually concurrent, a contradiction. Thanks to the total order of atomic multicast these deliveries occur in the same order in all correct nodes and the result is valid to these nodes.

In the second case, even if t_1 and t_2 were submitted to the DBMS at different nodes, their identifications are first atomically broadcast to all nodes, and the same reasoning of the precedent paragraph is also effective. Accordingly, the meta protocol implements *object access consistency*.

4.3 Handling the Global Phase Termination Event

The global phase termination event determines that a transaction was globally terminated at the nodes that took part in its execution. This event has to be launched by the replication protocol that is controlling the transaction execution, and the launch process depends on the implementation of the replication protocols. As examples, we consider here three implementation solutions, each based on a distinct termination scheme: distributed atomic commit, group communication, and reconnection of mobile nodes.

Informally, a distributed atomic commitment [Bernstein *et al.*, 1987] ensures that either all correct nodes that participated in the execution of the transaction commit its execution, or all of them will abort it. A protocol that implements a distributed commit involves all non faulty nodes, and should eventually terminate its execution with a common decision. A protocol based on consensus could also be used to reach such a decision [Guerraoui and Schiper, 1995]. In this case, the protocol effectively commits (or aborts) the transaction at the node after the decision of the adopted termination algorithm, and then it launches the termination event.

When group communication primitives are used (*e.g.* [Kemme *et al.* 1998]), [Fritzke e Ingels, 2001]), the termination event is simply obtained by the delivery of a final message of the replication control protocol to all correct nodes. Depending on the delivery properties (reliability and/or order), this message can determine the global phase termination of the transaction. Protocols use to commit (or abort) the transaction just after the delivery of that message, and after this they can also launch the event so that the meta protocol can handle it.

When we deal with reconnection of mobile nodes, as it is the case of replication control protocols presented in [Holliday *et al.*, 2000], the termination event corresponds to the end of the synchronization that occurs during reconnection. This synchronization usually involves the exchange of updates issued at the mobile node, say node *N,* or at the node to which the mobile node reconnects. If the database is fully replicated, this

synchronization can occur progressively between the nodes that become connected to *N*. For any node that perceives *N* reconnection, the global phase terminates after the synchronization with *N*. After this synchronization, the global termination event can be launched and treated by the meta protocol.

4.4 Node Failures and Mobility

As defined in section 3, we assume that nodes only attach and detach replication control protocols while they are all connected to each other. In order to track node mobility, one could use a node (or process) group membership service of a virtually synchronous group communication sub-system [Birman *et al.*, 1991]. Informally, virtual synchrony ensures that all correct nodes agree on the set of nodes that could have abandoned a node group and this in an ordered sequence of installed group views. Membership (or view) changes are usually result of either explicit requested *leave group* and *join group* operations or by involuntary node failures. Accordingly, mobile nodes may smoothly disconnect by issuing a leave group and reconnect with a join group request.

Our meta protocol can be executed only after a complete membership reconfiguration: all disconnected nodes have to be (re)joined to the group. During this connection period, failures are handled by the virtual synchronous group communication sub-system. We do not implement automatic membership reconfiguration if a node fails after it has executed an explicit leave request. Thus the meta protocol liveness could suffer in such a failure scenario, unless some manual intervention is taken. We note however that the meta protocol safety is not risked in this case if the set of original nodes is compared to the current group view.

5. Implementation

The proposed meta protocol is being implemented in a adaptable middleware for mobile database applications. The Figure 6 sketches the dynamic adaptable architecture that is being implemented. The mobile replicated database application runs of top of the meta protocol that is responsible for the replacement of replication control protocols, through the previously defined attachment and detachment procedures. The meta protocol and the mobile application interact with building blocks for distributed programming, aspect oriented programming and database access.

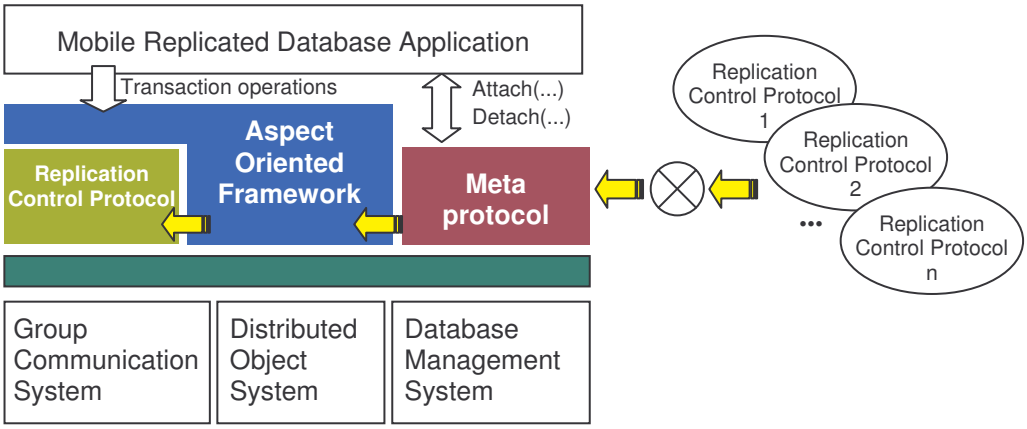

Figure 6 – The architecture for dynamic adaptable mobile replicated databases.

The group communication system offers atomic multicast and state transfer services, as well as a virtually synchronous execution environment. These functionalities are used by the meta protocol and can also be used by the replication control protocols.

State transfer is particularly useful for to the synchronization of database states on mobile node reconnections. The JGroups toolkit [Red Hat, 2006] has been used for that purpose. The distributed object system allows an abstract way to make remote method invocation among objects from the application and replication control protocols. Java RMI has been used here. The database management system provides the system with a transactional environment. We are using the MySQL DBMS.

The aspect oriented framework plays a central role in our architecture. Aspect orientation [Kiczales *et al* . 1997] has been proposed as a solution for the treatment of non functional and transversal issues in adaptable middleware [Gilani *et al.*, 2004]. Aspect oriented software enhances the modularity by separating orthogonal concerns in well defined constructions called *aspects.* Aspects can crosscut the static structure of software, adding for example a new method to a class, as well as the dynamic structure, for example by modifying the execution flow of a program. The points of interest in the execution of some program (also called *joinpoints*) are defined by declaring *pointcuts*. To the pointcuts are associated pieces of code called *advices* that can add new behavior to a program. Syntactically, *aspects* allow the definition of pointcuts and advices. According to the aspect oriented language or framework used, aspects can be *weaved* into the program code, either before the program starts or at run-time (dynamic weaving).

We have adopted, in the present implementation, the AspectWerkz framework [AspectWerkz, 2007]. AspectWerkz offers two important features: a dynamic weaver and a way to transform plain Java classes into aspects through a XML definition file. The first feature has a particular interest because it allows attaching the aspects code to some pre-programmed Java bytecode during the execution of the program. The second feature offers a way to decouple the definition of pointcuts (XML) from the actual implementation of aspects (Java classes). This brings flexibility to the definition of pointcuts and to the reuse of classes as aspects.

In our work, we consider that replication control protocols are concerns that should be separated from application programming, thus enforcing modularity and adaptability of the software development process. Accordingly, aspects are used to accommodate meta protocol functionalities. The aspect oriented framework enables the association of aspects to application and replication control algorithms. With respect to application, we intercept transaction operations as they are submitted to a DBMS. In practice, these points correspond to popular mechanisms for DBMS connection initialization, data access, connection interruption, like the methods offered by well known database programming interfaces (*e.g.* JDBC API, Java Database Connectivity). We have also to define the treatment of replication control protocol events, as the start of a transaction and its global termination. These operations and events can be declared in the scope of pointcuts that identify some application and replication control joinpoints. It is worth to note, that with the aspect oriented framework, we reach a complete decoupling from application, replication control and aspect code. These code components (Java classes) can be combined through XML elements defining pointcuts and advices.

We are implementing the dynamic architecture by extending a system that offers a set of adaptable replication services using an aspect oriented language [Tavares *et al.*, 2006]. The adaptability provided by the system in [Tavares *et al.*, 2006] allows the attachment of aspects written in the AspectJ [AspectJ, 2007] language to centralized database application written in Java and that use the JDBC API. The static configuration of aspects allows the extension of a client-server database with replication control

protocols suited to mobile environments. Depending on the aspects chosen by the application programmer, different types of replication can be implemented as for example, full replication and partial replication. The partial replication model allows also different types of data accesses to mobile nodes. The protocols implemented through aspects have been used as components of the dynamically adaptable architecture and will be used to validate the meta protocol in practical applications.

Kienzle and Guerraoui argue in [Kienzle e Guerraoui, 2002] that aspect oriented languages are limited in their utility when it comes to separate transactional mechanisms from the functionalities of applications classes. The main issue presented by the authors is that in the general case, it is impossible to completely decouple the semantics of application classes from the semantics of transaction mechanisms implemented by aspects, being only possible a "physical" separation of these components. That problem is associated to exception handling, method semantics, restrictions on transaction synchronization and the maintainability of the application, among other questions. However, the middleware we propose avoid most of these problems because it connects aspects to the JDBC API, that in contrast to the general case, offers a stable and well defined method interface, and thus allows a stable definition of pointcuts.

6. Concluding Remarks

This paper presented a meta protocol for the dynamic replacement of replication control protocols in adaptable mobile replicated databases. In best of our knowledge, the work that is closely related to ours is the meta protocol proposed in by Yang and Li [Yang and Li, 2004]. Yang and Li propose a meta protocol that allows to change concurrency control in collaborative applications, but that is not well suited to transactional systems. Also, they work relies on a central server that coordinates protocol replacement, while we assume a symmetrical protocol based on group communication, and so, without a single point of failure.

The adaptability of replication mechanisms has been proposed in previous work. Some systems rely on reflective models (*e.g.* [Kleinöder and Golm, 1996], [Fraga *et al.*, 1997]) and other on frameworks with specific components (*e.g.* [Drapeau *at al.*, 2002], [Beloued *et al.*, 2005]). These approaches allow, in different ways, the adaptation of the structure of replicated systems according to different distributed service needs. Our protocol is complementary, in the sense that it shows a way for implementing a dynamic replacement of replication protocols, potentially able to be included in other replication architectures. This work also differs from previous ones in that it explores aspect orientation as a mechanism to implement adaptability of mobile replicated databases.

7. References

AspectJ (2007). The Aspect Programming Guide. <http://www.eclipse.org/aspectj/doc/released/progguide/index.html>. Accessed in April, 2007.

AspectWerkz (2007). AspectWerkz - Plain Java AOP. <http://aspectwerkz.codehaus.org>. Accessed in April, 2007.

Beloued, A., Gilliot, J.-M., Segarra, M.-T., André, F. (2005). Dynamic Data Replication and Consistency in Mobile Environments. In: *Proc. of the 2nd. International Doctoral Symposium on Middleware*. Nov. 28 – Dec. 02, 2005. Grenoble, France. ACM.

Bernstein, P., Hadzilacos, V., Goodman, N. (1987). Concurrency Control and Recovery in Database Systems. Addison Wesley.

Birman, K., Schiper, K., Stephenson, P. (1991). Lightweight Causal and Atomic Group Multicast. ACM Transactions on Computer Systems, vol 9, no. 3, august, 1991.

Drapeau, S., Ronancio, C. L., Déchamboux, P. (2002). RS2.7: and Adaptable Replication Framework. In: 18èm. Journées de Bases de Données Avancées , Oct., 2002. France.

El Abbadi, A., Toueg, S. (1986). Availability in Partitioned Replicated Databases. In: *Proc. 5th. ACM-SIGACT-SIGMOD Symp. on Princip. of Datab. Syst.* ACM.

Fraga, J., Maziero, C., Lung, L. C., Loques Filho, O. (1997) Implementing Replicated Services in Open Systems Using a Reflective Approach. 3rd International Symposium on Autonomous Decentralized Systems (ISADS '97). Berlim. 1997.

Fritzke Jr. , U., Morselli Jr., J. C. de M., Abrão, I. C., Luiz A. Gomes, Faria, C., Vicentini, W. B. (2004). A Protocol for Mobile Replicated Databases based on Causallity. In: *Proc. Of the WSCF 2004*. Fortaleza, CE, 2004. SBC.

Fritzke Jr., U., Ingels, P.. Transactions on Partially Replicated Data based on Reliable and Atomic Multicasts. Proc. of the ICDCS-21. Phoenix-AZ, USA. April 16-19, 2001.

Gilani, W., Naqvi, N. H, Spinczyk, O. (2004). On Adaptable Middleware Product Lines. In: *Proc. of the 3rd Work. on Adapt. and Refl. Middl...* ACM. Toronto, Canada.

Gray, J., Helland, P., O'Neil, P., Shasha, D. (1996). The Dangers of Replication and a Solution. In: *Proc. of the 1996 ACM SIGMOD'96*. Montreal, Canada.

Guerraoui, R., Schiper, A. (1995). The decentralized non-blocking atomic commitment protocol. In.: Proc. of the IEEE Int. Symp. On Parallel and Distributed Processing (SPDP'95). San Antônio, Texas, EUA, September, 1995.

Hadizlacos, V. and Toueg, S. (1993). Fault-tolerant broadcasts and related problems. In Sape Mullender, editor. *Distributed Systems,* chapter 5. Addison Wesley, 1993.

Holliday, J., Agrawal, D., El Abbadi, A. (2000) "Planned Disconnections for Mobile Databases". Tech. Rep. TRCS00-07. Dep. of CS, UCSA, EUA. 2000.

Kemme, B., Alonso, G. (1998). A suite of database replication protocols based on group communication primitives. In: Proceedings of the 18th IEEE Int. Conf. on Distr. Computing Systems (ICDCS). pp. 156-163. Amsterdam, The Netherlands. 1998.

Kleinöder, J. and Golm, M. (1996). Transparent and Adaptable Object Replication Using a Reflective Java. Tech. report TR-14-96-07. CS Dept., Friedrich-Alexander University/Erlangen-Nürnberg University. 1996.

Kiczales, G.; Lamping, J.; Mendhekar, A.; Maeda, C.; Lopes, C. V.; Loingtier, J.-M.; Irwin, J. (1997). Aspect Oriented Programming. In: *Proc. of the ECOOP'97*. Springer.-Verl.. Jun., 1997.

Kienzle J., Guerraoui, R. (2002) AOP: Does it Make Sense? The Case of Concurrency and Failures. In: *Proc. of ECOOP 2002*, Malaga, Spain, Jun, 2002. LNCS 2374 /2002.

Red Hat, 2006. Red Hat Inc. (2006). Reliable Multicasting with the JGroups Toolkit – Revision 1.6. Available at <www.jgroups.org>.

Shapiro, M., Rowstron, A., Kermarrec, A.-M. (2000). Application-independent reconciliation for nomadic applications. In: *Proc. of the SIGOPS Europ. Works..: "Beyond the PC: New Challenges for the Oper. Syst.."*. Denmark. Sep. 2000.

Tavares, T. C., Gomes, L. A. F., Fritzke Jr., U., (2006) Middleware Adaptável para Bancos de Dados Móveis utilizando Aspectos. In: Anais do Work. de Desenvolv. de Software Orientado a Aspectos (WASP'2006). Florianópolis, SC. SBC.

Wiesmann. M. , Pedone, F. , Schiper, A., Kemme, B., Alonso, G.. Database Replication Techniques: a Three Parameter Classification. In: Proc. of SRDS'2000. pp. 206-215, Nürnberg, Germany, October 2000. IEEE.

Yang, Y.; Li, D. (2004) Separating data and control: support for adaptable consistency protocols in collaborative systems. In: *Proc. of CSCW'2004*, Chicago, USA. ACM.