ISSN (Print) : 2319-5940
ISSN (Online) : 2278-1021

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 2, Issue 3, March 2013

Business Logic and Long Lived Transactions Processing

Romani Farid Ibrahim

High Institute of Computer Science and Information - City of Culture and Science- 6 October City, Egypt.

Abstract: Many modern business applications are working as long lived transactions (LLTs) which should maintain database consistency to be a valid transaction. LLT models usually based on using compensating transactions, and many papers claimed that compensation process doesn't reserve database consistency. In this paper, we concentrate on disconnection and consistency of mobile transactions as an example of long-lived transactions. We extended the M-Shadow technique to handle both atomic mobile transaction applications, and transactional workflow applications with or without compensation and maintain database consistency. M-Shadow uses the notation of actionability and it is an optimistic concurrency control technique. It increases the transaction success probability even with disconnection and raises the performance of the system.

Keywords: Concurrency Control, Mobile Database, Transaction, Workflow, Shadow paging, Saga, Caching, Compensation.

I. Introduction

The values of all information systems are based on the accuracy and consistency of their databases. Accessing data anywhere-anytime-anyway it becomes real events, but this should not violate database consistency. The mobile database, or embedded database on a mobile device, is starting to become an important player in all practical fields, for example, business, traveling, police, military, medical, etc. The data is entered approximately in its real time, no delay between the events time and the entering time to the database. Also many modern business applications are working as long lived transactions (LLTs) which are transactions hold on to database resources for relatively long periods of time, significantly delaying the termination of shorter and more common transactions [20]. LLT to be a valid transaction should maintain database consistency.

As an example of long lived transactions, we concentrate on mobile transactions, which is a transaction performed with at least one mobile host takes part in its execution [21]; also, it may be defined with perspective of its structure as a set of relatively independent (component) transactions, which can interleave in any way with other mobile transactions [8].

As an example of applications that can be applied as long lived transactions, we are considering mobile hosts are laptop computers belonging to members of a big salespersons team. The salesperson performs a transaction that handles a customer big order which consists of groups of independent sub-orders or groups of dependent sub-orders which may include partially dependent sub-orders.

There are many types of transactions that are related to the subject of LLTs, we mention some of them that are related to our work as flat transactions, compensating transactions, contingency transactions, nested transactions, saga transactions, vital and non-vital transactions.

Flat transaction (or transaction) is defined as a means by which an application programmer can package together a sequence of database operations so that the database can provide a number of guarantees, known as the ACID (Atomicity, Consistency, Isolation, and Durability)[18]. Nested transaction is a collection of related subtasks, or subtransactions, each of which may also contain any number of subtransactions as a tree structure and only the leaf-level subtransactions are allowed to perform the database operations [15].

A compensating transaction is a transaction with the opposite effect of an already committed transaction. It is intended to undo the visible effects of a previously committed transaction, e.g., cancel car is the compensating transaction for rent car. A contingency transaction is invoked upon the occurrence of some failure condition and before commit of the transaction for which it is an alternative. It is intended to accomplish a similar goal as the original transaction, as opposed to the compensating transaction which is intended to undo a committed (sub) transaction [19]. A saga is a long-lived transaction that consists of a set of relatively independent subtransactions associated with them their compensating subtransactions. To execute a saga, the system must guarantee that either all of the subtransactions in a saga are complete or any partial execution is undone with their compensating subtransactions [20]. A vital transaction is a transaction that must be executed successfully (i.e. it has to commit) for its parent transaction to commit. A non-vital transaction may abort without preventing the parent transaction from committing [19].

Workflow is a collection of tasks organized to accomplish some business process (e.g., processing purchase orders over the phone, provisioning telephone service, processing insurance claims). A task can be performed by one or more software systems, one or a team of humans, or a combination of these [25].

We view a transaction as a program in execution in which each write-set satisfies the ACID properties [3], and the program that updates the database as a three folds module (phases): reading phase, editing phase, and validation and write phase. The main questions we attempt to answer in this paper are: 1- if the data on the primary server has been changed while the mobile unit (MU) is disconnected or working offline, how can the transaction continue its work? 2- What are the effects of business logic on the transaction structure? 3- If the compensation is used how the database consistency is achieved?

We extended the M-Shadow technique which is described in detail in [1], [2],[3] to be suitable for different mobile transaction applications according to the nature of business logic. M-shadow technique is an optimistic concurrency technique constructed on the shadow paging technique that is used in deferred database recovery and other OS techniques. Shadow paging technique uses two copies of data items, the shadow copy (original), and the edited copy (current). When a transaction commits, the edited copy becomes the current page, and the shadow copy is discarded, otherwise, the edited copy is discarded and the shadow copy is reinstated to become the current page once more.

The rest of this paper is organized as follow: Section 2 describes the related work. Section 3 describes the important points we considered to propose the extended model. Section 4 introduces the extended M-Shadow technique for atomic transaction applications, and for transactional workflow applications with and without compensation. Section 5 describes a summary of the implementation and performance of the proposed technique and the last section 6 concludes the paper and followed by the references.

II. Related Work

Most of the work handling mobile transactions as (Kangaroo [6], Moflex [9], Promotion [7], Reporting and Co[8], Escrow techniques [22], etc.) assume that the handoff process is under the mobile support station (MSS) responsibility [10], and the mobile support stations has the capability to transfer control and transaction history among servers while handoff procedure as [6], [8], [9]. However, this approach has many limitations, such as, if the mobile unit moves relatively slow such that the probability of the commitment protocol terminating at the same cell is high. If it is fast moving then a frequent migration of the control may increase the protocol latency and thus its vulnerability [10]. In addition, if a big number of MUs move among cells, so that most of the response time is spent in transferring data among cells.

Most of the used methods apply the concept of compensation and many paper claimed that compensation does not reserve database consistency [11] [12].

Most of the papers assume rarely changing data (Insurance data, Patients data, etc); the mobile unit has replica or caching subsystem. And the mobile replica is logically removed from the master copy of the object and is only accessible by the transaction on the mobile unit [23], so that they do not consider the case of changing data on the primary server while the transaction processing. In addition, they assume long disconnection or working offline and do not consider short disconnection case.

III. Important Considerations

In this section, we describe the important points we considered to extend M-Shadow model to handle transactional workflow mobile applications which are: motivating example, transactions and grouping, compensation and business logic, implementation of compensation, the effects of attributes types on the transaction behavior (actionability), and description of validation test.

A. Motivating Example

As an example of applications that can be applied as atomic mobile transaction application or as a transactional workflow mobile application, we are considering mobile hosts are laptop computers belonging to members of a big salespersons team. The salesperson performs a transaction that handles a customer big order which consists of groups of independent sub-orders, or groups of dependent sub-orders which may include partially-dependent sub-orders.

Figure 1 shows nine independent subtransactions grouped in three independent groups, they represent nine unrelated items of three sub-orders in a compound transactional workflow.

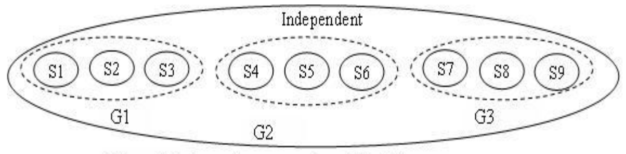

Figure 1 Independent transactional Workflow

ISSN (Print) : 2319-5940
ISSN (Online) : 2278-1021

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 2, Issue 3, March 2013

Because the nine items are unrelated, there is no effect of any subtransaction on the previous committed subtransactions, so that there is no need for using compensation in this case, and if any number of subtransactions or groups fails the remaining groups can continue their work.

Figure 2 shows the nine subtransactions as three dependent groups, this means that if any subtraction fails (except the doubly circle subtraction S5 because it is a non-vital subtransaction) the entire compound transaction should fails.

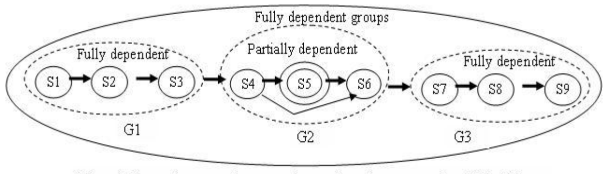

Figure 2 Dependent atomic transaction or dependent transactional Workflow

We view this case of compound transaction according to the business logic in two views: as atomic transaction and there is no need for compensation, or as a transactional workflow which will be similar to the saga model and should compensate for the previous committed subtransactions.

Figure 3 shows the nine subtransactions as three dependent groups but the doubly circle group2 is a non-vital group, this means that the compound transaction can commit without group2 or with group2, but no subtransaction of group2 can commit out of its group.

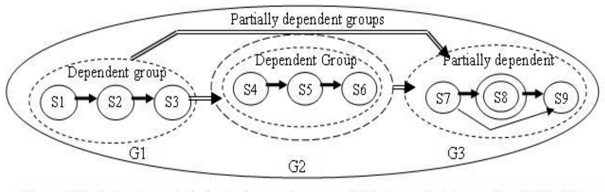

Figure 3 Partially-dependent atomic transaction or partially dependent transactional Workflow

Also according to the business logic, we view this case in two views: as atomic transaction or as a transactional workflow.

B. *Transactions and grouping*

Transaction is defined as a means by which an application programmer can package together a sequence of database operations so that the database can provide a number of guarantees, known as the ACID (Atomicity, Consistency, Isolation, and Durability)[18].

We view a transaction as a program in execution in which each write-set satisfies the ACID properties [3], and the program that updates the database as a three folds module (phases): reading phase, editing phase, and validation and write phase. We classified transactions based on their structures as simple transactions and compound transactions[5]. Simple transaction is a transaction that can not be divided into subtransactions and all ACID properties are achieved. Compound transaction consists of two or more simple transactions (called subtransaction) and theses subtransactions may be nested, it can be ACID or non-ACID. Examples of compound transactions are nested transactions, sagas, long duration transactions (LLT), kangaroo transaction, etc.

Simple transaction by nature is independent, but when it is grouped with other subtransactions in a compound transaction (CT), it has three cases:

- It does not lose its independency property, so it can commit alone.
- It loses its independency property, and it has a dependency relationship with its CT. IF it fails, the CT fails, if the CT fails for any reason, the subtransaction fails also.
- If it is a non-vital subtransaction, it can abort alone and doesn't effect on vital subtransactions of the CT and the CT can commit without it.

By this analysis we view that the independent transactions of saga model are lost their independency because they are grouped in a saga compound transaction and a dependency relationship is established among them, so if one transaction fails the entire saga should fail and all previous committed transactions should be compensated for their effects on the database.

In this paper we classified applications according to the division of their compound transaction to atomic compound applications which their compound transaction isn't divisible and satisfies ACID properties, and, transactional workflow (TW) applications which their compound transaction is divisible and satisfies semantic ACID.

C. *Compensation and Business Logic*

A compensating transaction is a transaction with the opposite effect of an already committed transaction. It is intended to undo the visible effects of a previously committed transaction, e.g., cancel car is the compensating transaction for rent car. But many papers claimed that compensation doesn't reserve database consistency [11],[12]. For example, suppose that the account initially has $X, and then a withdrawal transaction of $Y (where X >=Y) is executed and that the transaction will be compensated later. If another transaction commits applying an interest rate on the balance before the compensation has been performed (i.e. when the account has $(X-Y). The interest transaction was applied on a kind of dirty data, and therefore database consistency will not be preserved.

We view that compensation process can be an acceptable solution if it doesn’t contradict with business logic that will be evaluated as reserving of database consistency. Most researchers assume that compensating transactions will be written by the programmer who writes the application, which means that the programming process will be doubled, because for every function there is a compensating function, so that this solution seems to be not a good solution. For simplicity, we assume that compensation can be done on numeric attributes or text attributes (non-numeric) and then we will generalize for any data type.

1) *Compensation for numeric attributes*

Whatever the equations that will be applied on the numeric attribute it will generate a numeric value that changes the state of the attribute by increasing or decreasing its value:

New value – old value = ± change value

We will store the change value (i.e. no recalculation for any equation) which is the final effect of the transaction on the attribute at commit time, not before image and after image as it is implemented in database systems logs. In this case, all numeric attributes will be additive, commutative and compensated attributes, if we apply the change values that are generated from the transaction and not the transaction logic. When the compensation process is started the change values will be added to the current values of attributes, it should be logically succeeded and doesn’t contradict with the business logic or the integrity constrains of the database system. Also compensation transaction is a transaction; this means that it should transform the database from consistent state to another consistent state.

2) *Compensation for non-numeric attributes*

Examples of this case are transactions that cancel reservation for rooms in a hotel, cancel renting a car, or cancel reservation for airplane tickets. Usually these transactions handle future events and are not additive or commutative, and are related to specific date. This means that any transaction happened after the reserving transaction (ex, for the rooms) is a valid transaction because it doesn’t violates the database constrains or business rules and doesn’t contradict with other transactions, because it is for different date. The compensation process in this case will be restoring the old image (state) for that period, and will be implemented as a new subtransaction. The logic will be:

```
If object.current_value =  object.transaction_value then
        restore before image (old_value)
Else call interrupt handler (for human interaction).
```

For example, for reserving a room in a hotel the values of attributes will be (O for old, N for new):
OState: *empty* Nstate: *busy* renter: *abc* from_date: *1-7-2013* to_date*: 8-7-2013*

After compensation process the values of attributes will be:
OState: *busy* Cstate: *empty* renter: *null* from_date: 1-7-2013 to_date: *8-7-2013*

If compensation is applied on text attributes; it can be applied on any other type of attributes. For example, it can be applied on image or sound attributes based on time stamp of data items.

From the previous analysis, we view that compensation can be applied on all data types for business applications, and the problem isn’t in the compensation process itself, but it is in the implementation of the compensation.

D. *Implementation of compensation*

Compensation is used in transactional workflow (TW) applications only, because the business logic of atomic transactional applications doesn’t need to use compensation. The compound transaction of a transactional workflow application is divisible and should be semantically ACID. We have two cases with or without compensation. Transactional workflow applications with compensation mean that the business logic accepts the compensation process without any logical error, we differentiate between two types of failures; integrity constrains failure and network failure. Integrity constrains failure means the business logic is failed and causes the entire compound transaction to abort and compensation process should be started. Network failure can be happened because of disconnection or any other reason, and causes the current subtransaction to be restarted at reconnection time. This is because the nature of compound transaction is divisible, this means that the subtransaction can fail and restart many times without causing abortion of its compound transaction. Transactional workflow application without compensation is a group of independent transactions. This means that every subtransaction is complete by itself and its success or failure doesn't depend or effect on any other subtransaction in the compound transaction.

For transactional workflow application with compensation we need to create a transactional workflow log which stores the change values of numeric attributes and stores the old values and new values for non-numeric attributes. The current programming languages that handle transaction processing should include procedures for handling transactional workflow as *begin-trans-work* and *end-trans-work.* The structure of the transactional workflow will be something similar to the following structure.

```
Begin_trans_work  name
```

Begin *t1*
End *t1*
………….
Begin *tn*
End *tn*
Commit_trans-work name // do nothing because each subtransaction is committed or
Abort_trans-work name // which means start compensating process by applying the changes that is stored in the transactional workflow log,
End_ trans_work name

The flowing examples show the content of the transactional workflow log for numeric and non-numeric attributes.

Trans_work_id: *TW5* trans_id: *t3* DB: *inventory* table: *sold-amount* object-id: *512* column: *qty* change_ value: *50*

Trans_work_id: *TW5* trans_id: *t3* DB: *inventory* table: *balance* object-id: *512* column: *qty* change_ value: *-50*

Trans_work_id: *TW6* trans_id: *t1* DB: *hotel-reservation* table: *reservation* object-id: *room2* old-value: *busy* new_ value: *empty*

In mobile transaction processing the transactional workflow log can be simulated by storing the old values and new values of data items as XML files until the end of the transactional workflow on the mobile unit. At the compensation process the change values can be recalculated by subtracting old values from new values and applied in reverse order.

For transactional workflow without compensation, the compound transaction is divisible and there is no relationship among its subtransaction, any subtransaction can abort without any effects on previously committed subtransactions. For example, assume according to the business logic, it is allowed for customer to return items of orders for any real reason (ex.; because it is defective item). This case handle compound transaction already implemented in the past, which differs in the meaning from reserving a room in a hotel in the future, but it doesn't differ in its effect on the database. The seller creates a new independent returning transaction which deletes the effects of the previous committed transaction on the database, i.e.; the seller compensates for the subtransactions of the defectives items only without any effect on the other subtransactions of the transactional workflow, this is because the independency relationship among subtransactions. This case differs from the saga model which requires compensation for the entire saga.

The compound transaction of atomic transactional application is not divisible by nature. The business logic generates logical errors if the compensation process is applied after some subtransactions have been committed. It should be implemented using regular concurrency control witch usually is based on locking techniques. Example of this type of applications is the interest transaction that is mentioned in section 3.3. Or it can be implemented as a transaction workflow with additional constrains on the transaction processing, by checking that the data item that will be changed is not participated in any pending transactional workflow. In this case, semantic ACID is achieved. So that, we need to create a pending table that keep track for current pending transactional workflows, its structure should include: TW-id, DB-name, table-name, record-id, and no need for storing attribute-name to decrease the search time in the pending table and increase the performance of the system. When a transactional workflow finishes it's processing (i.e.; reach the *End_trans_work* command), it should be removed from the pending table. But if the pending TW accesses shared data items, it will decrease the performance of the entire system.

E. Actionability and Transactions Behavior

In this section we review the M-Shadow technique for atomic transaction applications and its related concepts, which will be modified to handle transactional workflow applications. In M-shadow technique, transaction's validation is not tightly coupled to the eventuality of encountering modifications (done by other transactions) on the values of one or more of its data items. Transaction behavior at run time depends on some characteristics of its set of data items. We use the notion of actionability [1][2][3] to describe how a transaction behaves if a value-change is occurred on one or more of its attributes during its processing time and by other transactions. Other than Key attributes *(K)*, actionability classifies the data items used by a transaction into five types: change-accept, change-aware, change-reject, change-passing and location-time attributes.

Change-Accept *(A)*: Any attribute retrieved during the read phase to complete and explain the meaning of the transaction. If it is potentially changed (by another transaction) while the transaction is processing, it does not have any effect on the transaction behavior.

Change-Reject *(R)*: This type of attributes is subject of periodical changes (e.g., Currency values, Tax rates, etc.). The value of such attribute remains constant for long period. But once it is changed during the transaction life time (by another transaction), it affects severely the transaction behavior.

ISSN (Print) : 2319-5940
ISSN (Online) : 2278-1021

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 2, Issue 3, March 2013

Change-Aware *(W)***:** This type of attributes is subject to change more frequently by different concurrent transactions. A modification on the value of this type of attributes may be accepted if the new value still in the acceptance range. Otherwise, the transaction aborts.

Change in				Integrity Constrains Violation	T Succeed
Change-Accept Attribute	Change-Reject Attribute	Change-Aware Attribute	Change-passing Attribute		▨
Y / N	N	N	Y / N	NA*	Y
Y / N	Y	N	Y / N	NA	N
Y / N	N	Y	Y / N	N	Y
			Y / N	Y	N

*NA means Not Available

Change-Passing *(P)*: this type of attributes is not basically part of the transaction data, but the result of the transaction processing is passed to this type of attributes. For example, in an insurance company (or many other applications) all different departments are related through the financial department, so that, all insurance transactions in all departments should pass their financial values to the financial attributes. Usually this subtransaction is succeeded because it only increases the financial attributes by the new amounts and the previous change and the current values of this type of attributes doesn't effect on the transaction data or behavior. But if the subtransaction that changes their values is failed for any reason, it causes the main transaction to fail.

Location-time *(L):* this type of attributes is for handling Location dependent transaction processing.

The previous three types of attribute actionability (Change-Accept *(A)*, Change-Reject *(R)*, and Change-Aware *(W)*) are to be declared for each transaction type. If omitted, the complete set of attributes will be handled as Change-Reject type (the default actionability type), a case in which the M-Shadow works like the traditional Shadow technique. Also, they are retrieved at the read phase to be edited and to apply the function of the transaction on it. It is also important to note that a transaction may generate a new data item *(G)* as a function of the three previous types of attributes. The M-Shadow technique handles these attributes exactly as before:

- If a Change-Reject attribute(s) is modified during the transaction processing, the complete transaction aborts.
- But else, if a Change-Aware attribute(s) is the modified attribute and the changes are within the acceptance ranges, the transaction is recalculated and continues, otherwise it aborts.
- But else, if a Change-Accept attribute(s) is the modified attribute, the transaction continues and writes values.

Table (1) illustrates the applied validation rules. If the Change-Accept attribute and the Change-passing attributes are changed or not, it doesn't have any effect on the transaction behavior that updates the Change-Aware attributes. Also, Change-Accept attributes are very rarely changing attributes, for example, item-description, employee-name; Birth-Date, etc., are approximately fixed value attributes.

Rule: If T1, T2 are concurrent transactions, T1 changes a shared Change-Reject attribute and T2 changes a shared Change-Aware attribute that belong to a normalized database then:

- If T1 commits before T2 then T2 must abort.
- If T2 commits before T1 then T1 can continue its processing.

The reasons behind using the actionability include:

- A transaction usually updates a part of the data set it uses, the other part of the data elements is asked by the transaction to control the transaction. These data items are read only items and a change in such elements should not prevent the execution of the transaction.
- Our concern is on the transactions that update Change-Aware attributes, which have acceptable range. An encountered change in these attributes may affect the outcomes of the transaction but not aborts entirely its execution.
- The usage of mobile transactions is still limited to salesperson and inventory applications which are, by nature, applying short transactions with little attributes. This fortunately complies well with the M-Shadow concept.

F. Description of Validation Test

The validation test for atomic transactions compares the original values of some data items with its current values on the primary server, which succeeds in three cases:

- No change, which means that the original values are equal to the current values on the primary server.
- Constrained change, which means that some Change-Aware attributes has been changed by other transactions during disconnection (working offline) time but still these changes within the integrity constraint acceptance range.
- Insignificant change, which means that some Change-Accept attributes has been changed by other transactions during disconnection (working offline) time or during the execution of the transaction, but these Change-Accept data items does not effect on the current transaction.

The validation test fails in the following two cases:

- Significant change, in which we detect that some Change-Reject data items have been changed during the transaction processing and/or disconnections.
- Out-of-Constraints change, in which we detect that one or more Change-Aware data items have been updated in such a way that the global changes put the stored values out of the acceptance ranges.

IV. THE M-SHADOW MODEL

This section describes the summery of the M-shadow technique for handling atomic mobile transaction applications and transactional workflow mobile applications with or without compensation. The validation- write procedure can be written as a part of the DBMS or as a stored procedures at the primary server side.

A. *Summary of the M-Shadow Technique Steps for atomic mobile transaction applications*

At Mobile Unit side:

1. Retrieve the current dataset from the primary server (Reading phase)
2. Copy the retrieved dataset as a shadow copy.
3. The user edits the dataset on the shadow copy [modify, add, delete] (Editing phase)

Begin write-set-transaction

4. Send the original read-set, the edited-set (shadow copy changes), the read-query, and the update query to the primary server (subtransaction by subtransaction).

At Primary Server Side:

5. Implement the validation and write phase (which can be implemented as a part of the DBMS or as a stored procedure at the primary server).
   - Call validation-write-1 procedure (as a part of the DBMS)
6. If one subtransaction fails (disconnection, integrity constraints, etc.)

At Primary Server Side:

- Rollback the current and all the previous write-set subtransactions of the group.

At Mobile Unit side: because of

- **Integrity constraints violation:** Drops its data-sets and clears the memory to start a new transaction.
- **Short disconnection:** Try to reconnect.

TABLE 2: TRANSFERRING AMOUNT FROM ACCOUNT X TO ACCOUNT Y.

Read-Phase: $K(X)$, $A(X)$, $W(X)_o$, $K(Y)$, $W(Y)_o$	10 , Abc, 5000 , 20 , 3000
Edit-Phase: $K(X)$, $A(X)$, G, $F_1(g) \rightarrow \Delta_{(W(X))}$ $\Delta_{(W(X))} + W(X)_o = W(X)_s$ $F_2(\Delta_{(W(X))}) \rightarrow \Delta_{(W(Y))}$ $\Delta_{(W(Y))} + W(Y)_o = W(Y)_s$	10 , Abc $F_1(5000) \rightarrow -400$ 5000-400 = 4600 $F_2(-400) \rightarrow 400$ 3000 + 400 = 3400 10,ashraf, 4600, 20, 3400
Validation and Write Phase: ***Validation Test for account X:*** Current Value at Primary Site: $K(X)$, $A(X)$, $W(X)$ $\Delta_{(W(X))} = W(X)_s - W(X)_o$ $W(X)_c = W(X)_c + \Delta_{(W(X))}$ If(check-constraints($W(X)_c$) then Accept $W(X)_c$, G Commit (t) Else Rollback (t) End if	 10,Abc, 7000 -400 = 4600 -5000 6600 = 7000-400 check-constraints(6600)= True Accept 6600 , F1(5000) , -400 Commit (t)
Validation Test for account Y: Current Value at Primary Site: $K(Y)$, $W(Y)_c$	 20, 2000
$\Delta_{(W(Y))} = W(Y)_s - W(Y)_o$ $W(Y)_c = W(Y)_c + \Delta(W_{(Y)})$ If(check-constraints($W(Y)_c$) then Accept $W(Y)_c$, G Commit (t) Else Rollback (t) End if	400 = 3400 -3000 2400 = 2000 + 400 check-constraints(2400)= True Accept 2400 , F2(-400) , 400 Commit (t)

- **Long disconnection:** The program saves the data-sets (the original data-set and the shadow data-set) as XML files on the mobile unit secondary storage.

When reconnection with the primary server is available

After short disconnection:

The program reissues the dependent-write-set group transaction as a new transaction as in step 4.

After long disconnection

The program loads the XML files and starts a new fully dependent write-set group transaction for the loaded data-sets (shadow and original) as in step 4.

Commit or abort **write-set transaction**

Validation-Write Procedure-1 (A General Validation

Algorithm to Be Put as a Part of the DBMS)

Validation-Write-Phase (Record original, Record shadow, String read-query, String update-query)

In what follows we show the core functions of the technique, which use the actionability rules to perform the validation test. Its inputs are original data-set, shadow dataset (shadow-rec), read-query, update query, and the actionability types for attributes if they are not declared while tables creation. If the validation test succeeds, the transaction commits, otherwise the transaction aborts.

Aware-Update (integer flag)

```
{
For each change-reject-attribute(i)  in shadow-rec
  If   Current.R(i) <>  Shadow.R(i)     then
     Flag = -1
     Goto par-out
  End if
Next-For

 For each change-aware-attribute(i)   in shadow-rec
    $\Delta$W(i)   = Shadow.W(i)   - Original.W (i)
    Current.W(i)   = $\Delta$W(i)  +  Current.W(i)
      If (check-constraints(current.W(i)   ) = False ) then
         Flag = -2
         Goto par-out
  Next-For
Par-out:
Return (flag)  }
```

Table 2 shows an example to describe how the validation and write phase can be applied, and assume linear transactions for simplicity. The example shows a bank transaction that transfers $400 from account X to account Y. We use the notations of actionability, K denotes the Key attribute, *A* denotes a Change-Accept attribute, *R* denotes a Change-Reject attribute, *W* denotes a Change-Aware attribute, *G* denotes a generated attribute, and the subindexes *o* denotes the original value, *s* denotes the shadow value and *c* denotes the current value at the primary sever.

B. M-shadow technique for transactional workflow mobile applications with compensation

Begin_ trans_workflow

At Mobile Unit side:

1. Retrieve the current dataset from the primary server (Reading phase)
2. Copy the retrieved dataset as a shadow copy.
3. The user edits the dataset on the shadow copy [modify, add, delete] (Editing phase)
4. Send the original read-set, the edited-set (shadow copy changes), the read-query and, and the update query to the primary server (subtransaction by subtransaction).

At Primary Server Side:

- Call validation-write-2 procedure (Stored Procedure at the primary server)

5. If one subtransaction fails because of disconnection:

At Primary Server Side:

- Rollback the current write-set subtransaction only.

At Mobile Unit side:

- Short disconnection (the user doesn't close the program which means all variables and data-sets still available in the main memory): Try to reconnect.
- Long disconnection (the user wants to close the program): The program saves the data-sets (the original data-set and the remaining elements of the shadow data-set) as XML files on the mobile unit secondary storage to be retrieved at the reconnection time.

When reconnection with the primary server is available:

After short disconnection:

The program resends the write-set data for the subtransaction, which the disconnection happened through its update only. The primary server restarts the write-set subtransaction as in step 4.

After long disconnection:

The program loads the XML files and starts a new independent write-set group transaction for the loaded data-sets (original and shadow) as in step 3.

End_trans_workflow

Validation-Write Procedure-2 (Stored Procedure at the primary server)

Sub Validation-Write (t_i)

```
 {
Begin write-set subtransaction ($t_i$)
   Hold exclusive lock ($t_i$)
      Read data from active database for ($t_i$) as current
      If  change-reject data-item is changed then
         Rollback transaction ($t_i$)
            Call compensation-handler
```

```
    Else
        Calculate $\Delta(x)$ = Shadow(x) - Original(x)
        Current (x) = Current (x) + $\Delta(x)$
        Check-validity (Current (x))

If check-validity success then
   Write shadow data-set to the current database
     Write change-values to TWlog
    //Or save shadow data-set as compensation-set and
original data-set it as XML files for compensation purpose.
Commit Trans ($t_i$)
  Removes the subtransaction shadow data-set from
the shadow copy}
    Else
          Rollback transaction ($t_i$)
           Call compensation-handler()
    End if
 End if
If network failure then
    At reconnection, restart subtransaction
End IF
        }
```

Compensation Handeler()

```
{
    For each commited-subtransactin ($t_i$) in TW-log
     Hold exclusive lock ($t_i$)

    For each changed attribute
      If attribute is numeric then
     Current_value = current_value+ change value
          If check-validity fails
            Rollback ($t_i$)
              Generate request report for human interaction
          Else
           Loop
        End if

    Else If attribute is non-numeric then
      If Current_value <> TW-log-value then
           Rollback ($t_i$)
   Generate request report for human interaction
     Else
        Current-value = old_value
          Loop
      End if
    End if
Write the current values to the database
Commit }
```

C. *M-Shadow technique for transactional workflow mobile applications without compensation*

Begin_ trans_workflow

The steps from 1 to 4 of TW with compensation are repeated.

5. Implement the validation and write phase:
   - Call validation-write-1 procedure (as a part of the DBMS)

6. If one subtransaction fails :

At Primary Server Side:

- Rollback the current write-set subtransaction.

At Mobile Unit side:

If failure because of:

- Integrity constrains:
  - Remove the subtransaction shadow data-set from the shadow copy.
  - Send next subtransaction data to the primary server.
- Short disconnection (the user doesn't close the program which means all variables and data-sets still available in the main memory): Try to reconnect.
- Long disconnection(the user wants to close the program): The program saves the data-sets (the original data-set and the remaining elements of the shadow data-set) as XML files on the mobile unit secondary storage to be retrieved at the reconnection time.

When reconnection with the primary server is available:

After short disconnection:

The program resends the write-set data for the subtransaction, which the disconnection happened through its update only. The primary server restarts the write-set subtransaction as in step 5.

After long disconnection:

The program loads the XML files and starts a new independent write-set group transaction for the loaded data-sets (original and shadow) as in step 4.

End_trans_workflow

D. *Advantages and Limitations of the M-Shadow technique*

The advantages of using the M-Shadow technique are:

1. Transaction structure is build according to the business logic.

2. Reserve database consistency in case of using compensation.
3. Increase the performance of the system, by increasing the success probability of transaction by allowing transaction to continue its work even after disconnection and changing data on the primary server.
4. No transfer of logs or transaction history among sites. Only external files (XML files) would be saved on the mobile unit and will be deleted when the transaction finished.
5. Recovery for active transactions at failure time, which DBMS recovery manager does not do.
6. Decrease the programming time for applications, because the DBMS performs the update process.
7. No need to load the mobile unit with DBMS, replica and synchronization of replica.
8. No storage lost on the primary server or on the mobile unit, because after the transaction committed or roll backed, the program deletes the XML files.
9. The primary server load would be more lite.
10. More control over the network disconnection, especially in wireless networks which its property is frequently disconnection.
11. All ACID properties are achieved in atomic mobile transaction applications, and semantic ACID is achieved in the transactional workflow mobile applications.

The limitations of the M-Shadow technique are: it is designed for commercial applications that have a few shared data items among transactions and the validation test is not suitable for some real-time applications.

V. SUMMARY OF IMPLEMENTATION AND PERFORMANCE EVALUATION

To evaluate the effects of using the actionability types and rules, we used the simulation program Benchmark Factory for Databases, but it does not allow changing data while the simulation process is running. So that, we developed a prototype for the M-Shadow model with and without actionability as an atomic mobile transaction application and as a transactional workflow mobile application, we stored the new values and old values of data items as XML files and then recalculated the change values from them at reconnection time or at compensation process,. We found that, in atomic transaction mobile application without actionability, the compound transaction that fails if one of its vital subtransactions fails because of any data change at the primary server; it succeeds when the actionability types and rules are applied.

In transactional workflow mobile application without actionability, the transaction that fails because of any data change at the primary server; it succeeds when the actionability types and rules are applied, that increases the number of succeed transactions and the success rate. Also, applying compensation is based on business logic and reserves database consistency.

When implanting an atomic mobile transaction application as transactional workflow mobile application based on TW-log and pending table, the throughout of the system is decreased.

We implemented a sales application that uses the M-Shadow technique using Visual Basic .Net and SQL Server 2005 because they support many new features as writing and reading XML files. We assume that the replication handling is solved as a distributed database problem using the lazy replication technique among fixed hosts.

VI. CONCLUSION

In this work, we classified business applications to atomic transaction applications and transactional workflow applications with/without compensation based on the nature of business logic with maintaining of database consistency. We concentrate on mobile transaction as an example of long-lived transactions and extended the M-Shadow technique to handle both atomic mobile transaction applications and transactional workflow applications. M-Shadow technique increases the transaction success probability even with disconnection, and this by consequence, raises the performance of the system. Actionability classifies the data elements handled by a transaction according to how much a change on these elements affects the transaction behaviour. Also, applying compensation is based on business logic and reserves database consistency

Future research will extend this work to support complex business applications that include a big number of shared data items and complex computations, and web service applications. Parallel processing, real-time environments, compensation for location based transactions, alternatives for handling failure of compensating transaction other than requesting human interaction and security of mobile transactions will be investigated.

REFERENCES

[1] Romani Farid Ibrahim, “Adaptive Mshadow model for Handling Mobile Database Transaction processing", International journal of Advanced Research in Computer and Communication Engineering (IJARCCE) , Volume 1 Issue 9, pp700–709, November 2012.

[2] Romani Farid Ibrahim, Handling Disconnection in Mobile Database Transaction, The 5th International Conference on Application of Information and Communication Technologies (AICT2011), Azerbaijan, Baku, 2011.
[3] Osman Hegazy, Ali El Bastawissy and Romani Farid Ibrahim, Handling Mobile Transactions with Disconnections using a Mobile-Shadow Technique, Proceedings of the 6th International conference of Informatics and Systems (INFOS 2008), Faculty of Computers & Information-Cairo University, March 2008.
[4] Osman Hegazy, Ali El Bastawissy and Romani Ibrahim, A Programming Solution for Moving Mobile transaction, Proceedings of the 6th International Enformatika (IEC 2005), Budapest, Hungary, October 2005.
[5] Osman Hegazy, Ali El Bastawissy and Romani Ibrahim, Technique for Handling Transactions that Move among Hosts in Mobile Databases, Proceedings of the International Conference on Computational Intelligence (ICCI 2004), Istanbul, Turkey December 2004.
[6] M. Dunham and A. Helal, A Mobile Transaction Model that Captures both the Data and Movement Behaviour, Mobile Networks and Application (MONET), pp149–162, 1997.
[7] Gary D. Walborn and Panos K. Chrysanthis, PRO-MOTION: Management of mobile transactions, Proceedings of ACM symposium on Applied computing April 1997.
[8] P. K. Chrysanthis, Transaction Processing in a Mobile Computing Environment, Proceedings of the IEEE Worskhop on Advances in Parallel and Distributed Systems, 1993.
[9] Yin-Huei Loh, Takahiro Hara, Masahiko Tsukamoto, Shojiro Nishio, Moflex Transaction Model for Mobile Heterogeneous Multidatabase Systems, Proceedings of the symposium on Applied computing, ACM, March 2000.
[10] Nadia Nouali, Anne Doucet and Habiba Drias, A Two-Phase Commit Protocol for Mobile Wireless Environment, Proceedings of the 16th Australasian Database Conference (ADC 2005), Australian Computer Society, vol. 39, pp135–143, 2005.
[11] A. Elmagarmid, J. Jing, J. G. Mullen and J. Sharif-Askary, Reservable Transactions: An Approach for Reliable Multidatabase Transaction Management, Technical Report SERC-TR-114-P, Software Engineering Research Centre, April 1992.
[12] Paul Greenfield, Alan Fekete, Julian Jang, Dean Kuo, Compensation is Not Enough, EDOC 2003.
[13] Tim Edmonds, Steve Hodges and Andy Hopper, Pervasive Adaptation for Mobile Computing, Proceedings of the 15th International Conference on Information Networking, 2001.
[14] M. Satyanarayanan, Fundamental Challenges in Mobile Computing, Proceedings of ACM Symposium on Principles of Distributed Computing, 1996.
[15] Thomas M.Connolly and Carolyn E.Begg, Database Systems- A Practical Approach to Design, Implementation, and Management", Addison Wesley1999.
[16] Jose Maria Monteiro, Angelo Brayner and Sergio Lifschitz, A Mechanism for Replicated Data Consistency in Mobile Computing Environments, Proceedings of ACM symposium on Applied computing, Seoul, Korea, pp 914-919, March, 2007.
[17] Vijay Kumar, Mobile Database Systems, John Wiley & Sons, pp 113- 197, 2006.
[18] Patrick O'neil and Elizabeth O'nell, Database Principles Programming Performance, Academic press 2001.
[19] Elmagarmid,A.K, "Database Transaction Models for Advanced Applications", Morgan Kaufmann Press, March,1992.
[20] Hector Garcia-Molina and Kenneth Salem, "SAGAS", Proceedings of the ACM SIGMOD International Conference on Management of Data, 1987.
[21] Patricia Serrano-Alvarado_, Claudia L. Roncancio and Michel Adiba, "Analyzing Mobile Transactions Support for DBMS", Proceedings of the 12th International Workshop on Database and Expert Systems Applications (DEXA'01) , 2001.
[22] Narayanan Krishnakumar and Ravi Jain, "Escrow techniques for mobile sales and inventory applications", Wireless Networks, August, 1997.
[23] Joanne Holliday, Divyakant Agrawal and Amr El Abbadi, Disconnection modes for mobile databases, Wireless Networks, July, 2002.
[24] Zaihan Yang and Chengfei Liu, Implementing a flexible compensation mechanism for business processes in Web service environment, International Conference on Web Services (ICWS 2006), IEEE Press, Salt Lake City, Utah, 753–760.
[25] Diimitrios Georgakopoulos, Mark Hornick and Amit Sheth, An Overview of Workflow Management: from Process Modeling to Workflow Automation Infrastructure, Distributed and Parallel Databases, Kluwer Academic Publishers, 1995.

BIOGRAPHY

Romani Ibrahim received the B.A. in computer and information system from Sadat Academy for Management Science, Egypt. M.Sc. degree in computer science and Ph.D. degree in information systems from Cairo University, Egypt. He works in the High Institute of Computer Science and Information - City of Culture and Science- 6 October City. Egypt. He is a member of ACM. His research interests include distributed and mobile database systems, transaction processing, data warehousing and information security.