Temporal Consistency in Nomadic Mobile Environment

Dmitry S. Kochnev *Tatiana Kochurova* *Alexander Slobodianik*
SmartPhoneLabs LLC *University of Saint-Petersburg* *University of Saint-Petersburg*
{dmitry.kochnev; tatiana.kochurova; alexander.slobodianik}@smartphonelabs.com

Abstract

This paper presents a model of a nomadic middleware system with support for temporal consistency of structures semantically associated to XML-documents. Specially defined high-level operations commute with each other in most cases reducing amount of transaction aborts and increasing system availability.

1. Introduction

We live in a mobile world. Mobile technologies have penetrated into all areas of our life. And business-to-employee (B2E) applications are among the most promising directions of the mobile IT industry.

Classical ACID-transactions are inapplicable for the mobile environment due to a number of restrictions imposed on mobile applications. Basic problems that arise when moving a system to the mobile environment are data synchronization (i.e. restoration of data consistency) and maintenance of availability (unsuitability of data locking) [1]. This happens since mobile devices have limited resources, and performance-communication characteristics of a cellular network may fluctuate depending on a place, time of day, weather, activity of subscribers and many other factors.

The highlighted problem cannot be solved in the general case. In this paper we propose a solution for a wide range of problems with some restrictions for data access paths. As most B2E-applications (e.g. sales force automation), the sytem in the scope of our research uses data that may be represented as a tree-like structure semantically associated to the graph stored in the XML document.

A middleware model, considered in the scope of this paper, has nomadic architecture and its clients do not communicate directly with each other. The paper relates to the implementation roadmap of the middleware for the mobile distributed system presented in [2].

The main objectives of the current research are full implementation of the temporal data model, creation of the corresponding set of high-level operations and adaptation of the protocol proposed in [2] for the considered environment.

2. Related Works

The approaches to adaptation of well known wired architectures for the mobile environment are presented in [3]. The middleware design approaches are resulted in [2]. The restrictions of mobile environment and mobile platforms such as limited sizes of memory, storehouse of the data and some methods of mobile applications development are presented in [4].

The XML-tree can be considered like a B-tree or similar indexing structures. There are some efficient concurrency control protocols on B-trees (such as linking) [5]. But anway these approaches could not be used here because of inapplicability of server's data locking during disconnected operation (when the mobile connection is lost). [1]

2.1. High-Level Operations

There are a lot of data models, which have been known for a long time, that use abstract high-level operations defined as sequences of atomic operations. The concepts of multi-level [6] and nested [6], [7] transactions and sagas [8] are most interesting for our purposes. In most cases specially defined high-level operations commute with each other independently of elementary operations laying in their basis. Use of similar approaches improves concurrency of transactional operation in the system.

Data considered in this paper may have non-numerical nature and complex internal structure, therefore some semantic conflicts arising between operations can't be represented as pseudo-conflicts, and thus universal commutability can't be achieved as in pseudo-conflicts between *Withdraw()* and *Deposit()* operations working with accounts [6]. The proposed solution is to define the operations so that the number of irresolvable conflicts is as low as possible.

In [9] and [20] ideas of concurrent video model related to high-level operations are applied to a linear data model. We admit that further development of these ideas will allow us to apply a similar approach for handling XML structures.

2.2. Temporal Data Models

Motivation and fundamental knowledge about temporal databases are available in [10], [11] and [12].

An overview of temporal entity-relational models is presented in [13]. There are some popular approaches to temporal data representation. The problem is that none of these approaches is suitable for real life due to the complexity of its mapping to classical ER-model or directly to SQL database.

Works [14] and [15] present a model extending the system with temporal database features, and highlight problems related to the current research, such as data availability and concurrent execution of transactions. The authors propose a set of new query operators, needed in order to support an XML query language which supports temporal operations. In [15] algorithms for execution of temporal XML queries, based on query operators introduced in [14], are described and implemented using a query language which is based on a mix of Lorel, Xyleme query language, and elements of XPath and XQuery.

In [16] it is showed that valid-time, transaction-time and bitemporal databases can be naturally managed in XML using temporally-grouped data models. The technique is general and can be applied to historical representations of relational data, XML documents in native XML databases, and version management in archives and web warehouses. The issue of choosing values for the timestamps is covered in [17].

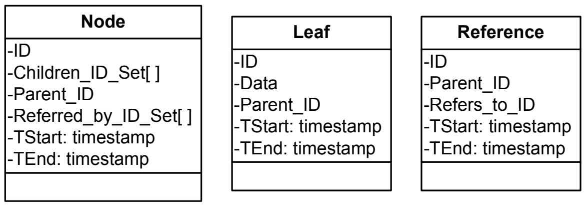

Figure 1. Node Structure.

The problem of representing in XML context-dependent data is highlighted in [18] and related works are available at http://www.ionio.gr/~manolis/, where an extension of XML, namely Multidimensional XML is proposed.

3. Data Model

Data used in the proposed model may be represented as a tree-like structure semantically associated to the graph stored in the XML document. It allows sophisticated manipulations due to different node levels, hierarchy among nodes, and the relationships among different elements, which could be defined.

Data used in the internal representation of the proposed model is taken out from the normal XML by applying the transformation, which moves the attributes into the affiliate nodes. Thus, the target XML has the following properties:

- Non-leaf nodes contain only navigational information.
- Data values are stored only in leaf-nodes.
- References are stored only in leaf-nodes. Each reference-node points to no more than one node. Also, reference-nodes point to navigational nodes only.

The proposed model uses transaction time being additional dimension of the structure storing the data. It is implemented in additional attributes *TStart* and *TEnd*. The first attribute contains the timestamp of the creation of the node, i.e. the moment starting from which the node became valid. The second one indicates the time of its removal, i.e. the moment starting from which the node became non-valid. If the node is valid, *TEnd* is set to a special constant *NOW*, always being interpreted as a current moment. Besides that, each node is associated with the identifier *id*, being a unique identifier of this node in the snapshot for each time moment. Thus, each node may be identified by a pair of *(id; TEnd).*

The internal structure of the temporal XML nodes is shown in Figure 1. A brief description of the properties follows:

- *ID* is an identifier of the node; it is unique for each node of the source (non-temporal) XML graph
- *Children_ID_Set[]* is an array of identifiers of the child nodes. Note, that each identifier is a pair of *(id; TEnd)*
- *Parent_ID* is an identifier of the parent node, represented as a pair of *(id; TEnd)*
- *Referred_BY_ID_Set[]* is an array of the identifiers *(id; TEnd)* of the reference-nodes pointing to the current node
- *TStart*, *TEnd* are the timestamps identifying the interval of validity of the node
- *Data* is a property of a data node containing data entry associated with this XML-node

- *Refers_to_ID* is a property of a reference node containing a pair of *(id; TEnd)* of the referred to node

4. Operations

Each transaction (being a sequence of operations) has a timestamp defined as a timestamp of the start of the transaction. The proposed solution defines the following operations on the tree-like data structure:

Definition 1. The insert procedure *Insert([id], parent_id, [sibling_id]): id* inserts a new child of the node identified with (*parent_id*; *NOW)* after the node identified with *sibling_id,* sets *TStart* of the created node to the *transaction timestamp*[1], set its *TEnd* to *NOW.*

Definition 2. The insert operation for navigation nodes *InsertNode([id],parent_id, [sibling_id]): id* inserts a new navigation node in the data tree using *Insert(id, parent_id, sibling_id)* procedure.

Definition 3. The insert operation for data nodes *InsertData([id], parent_id, [sibling_id], data): id* inserts a new data node in the tree using *Insert(parent_id, [sibling_id])* and associates the created node with *data* entry.

Definition 4. The insert operation for reference nodes *InsertRef([id], parent_id, [sibling_id], toID): id* inserts a new reference node in the tree using *Insert([id], parent_id, [sibling_id])*, associates the *Refers_to_ID* property of the created node with the identifier *toID* and inserts the identifier of the created node into the *Referred_by_ID_Set* of the found node with identifier *toID.*

Definition 5. The update operation for data nodes *Update(id, new_data)* updates data entry in the node with the identifier *(id*; *NOW)*, i.e. it sets *TEnd* of the found node to the *transaction timestamp* and inserts the new data node with the value *new_data* as a child of the node with identifier *(parent_id*; *NOW).*

Definition 6. The delete operation *Delete(id)* removes the full sub-tree with the root node identified as *(id*; *NOW)* and all reference nodes pointing to the nodes of this subtree, i.e. it sets the *TEnd* of all removing nodes to *transaction timestamp*.

Definition 7. The move operation *Move(id, new_parent_id, new_sibling_id)* moves all valid nodes of the sub-tree with the root node identified by *(id*; *NOW)* to the sub-tree with the root node being the child of the node *(new_parent_id; NOW),* following after the node with the (*new_sibling_id; NOW)* identifier, i.e. deletes the old subtree using *Delete()* and creates the new subtree containing only the valid nodes and appropriate valid references using *Insert*()*, the identifiers of the nodes in the new tree are the same as in the old one.

Definition 8. The select operation *Select(id, time)* returns a node with the identifier *id*, $TStart \leq time$ and $TEnd > time$ as an object, or reports on failure. The definition area of this operation is $t \in \{t \mid t > T\}$, where *T* is a *transaction timestamp.*

Definition 9. The select operation for current nodes *SelectCurrent(id)* returns a node as an object with identifier *(id*; *NOW)* or reports on failure.

Definition 10. Two operations $op_1(X; t_1)$ and $op_2(Y; t_2)$ **commute** if for each timestamp *t'* operations $op_1(X; t')$ and $op_2(Y; t')$ commute, i.e.:

- The order of execution of these operations has no influence on the final result
- The order of execution of these operations has no influence on its output values.

Table 1. Commutability table.

	IN	ID	IR	U	D	M	S	SC
InsertNode	+	+	+	+	+	+	0	A
InsertData	+	+	+	+	+	+	0	A
InsertRef	+	+	+	+	+	+	0	A
Update	+	+	+	0	+	+	0	A
Delete	+	+	+	+	+	+	0	A
Move	+	+	+	+	+	+	0	A
Select	0	0	0	0	0	0	A	A
SelectCurrent	A	A	A	A	A	A	A	A

Table 1 shows the advantages of the proposed approach above the elementary read and write operations. The symbol '+' means that the pair of corresponding high-level application commutes between theirselves better than the low-level operations laying in their basis. The symbol '0' means that there's no additional advantage. The symbol 'A' means that in both cases operations commute for all definition area.

Table 2. High-level and Low-level Operations.

Update(id) ~	InsertNode(id, pid, sid) ~
~ r(id.PARENT_ID, NOW) •	~ r(pid, NOW) •
• r(id, NOW) •	• r(sid, NOW) •
• w(id.PARENT_ID, NOW) •	• r(id, NOW) •
• w(id, T*) •	• w(pid, NOW) •
• w(id, NOW)	• w(id, NOW)

For example, *InsertNode(id_1, pid, sid)* and *Update(id_2)* commute in all cases in spite of the elemental read and write operations laying in their basis. The details are available in Table 2.

[1] *Transaction timestamp T** is defined as a moment of the current transaction start.

5. Protocol

As it was discussed above, ACID-transactions are not suitable for the mobile environment. The proposed solution uses AC-transactions to improve the availability of the system. An AC-transaction avoids isolation statement utilizing the local commitment of transactions at the client side. The results of execution of locally committed transactions become available for other transactions at the client side. An AC-transaction also avoids the durability statement because the server may abort some locally committed transactions.

The client manages two replicas of data. The primary replica is identical to the master replica stored on the server at the moment of the last connection to the server. All local changes are applied to the tentative replica.

The exchange protocol applied between client and server parts is fully inherited from [2]. The large scale of the protocol is shown in Figure 2.

First, the client subscribes for the data and receives from the server a replica marked with a special CRTS timestamp. The Client Replica TimeStamp (CRTS) is the last moment of time when the client's primary replica is identical to the (server's) master replica, e.g. it may be implemented as a moment of closing the client's connection. When disconnection occurs, the client continues working in disconnected mode with its replicas of data.

At some moment depending on the third party application, the client restores the connection, processes the general local commitment and sends the changes of its replica to the server. A general local commitment is a local commitment of all transactions. The server serializes the received transactions according to the Reconciliation Algorithm (RA) derived from the MTO (Multiversion Timestamp Ordering) protocol [7]. It states that the transaction is seralizable i.e. it may be added to the history if the following statement is true for each operation $op(y_1, \ldots, y_m, t)$ of the transaction with the timestamp t:

$$\forall op_i(x_1 .. x_n, t_i) \in H \mid t_i > t:$$
$$op_i \circ op == op \circ op_i$$

The server adds the serialized transactions to the end of the history and sets the timestamps of its operations to the transaction time being equal to the actual time of processing of the transaction on the server side. The policy of transaction aborting may be defined on the application side. For example, it is possible to abort a transaction with the minimal timestamp.

Next, the server sends to the client all transactions modifying the client's replica after the CRTS, the list of aborted local transactions and a value of the new CRTS. The client synchronizes its local time, i.e. if the time value at the end of connection is less than the received CRTS then the current time sets to the CRTS. Then the client applies the received changes to the primary replica, copies primary replica to the tentative one and tries to replay all transactions that have been committed locally after the last general local commitment according to the RA.

The formal proof of the correctness of this protocol is represented in [2].

If the client application uses semantic time as a *data* parameter, e.g. *InsertData(parent_id; getTime())*, such operation will be reconciled to a server history as *InsertData(parent_id; CLIENT_TIME + D)*, where *CLIENT_TIME* is a result of execution of *getTime()* at the client side, and *D* is a difference between time values at the server and at the client sides calculated during client-server interaction.

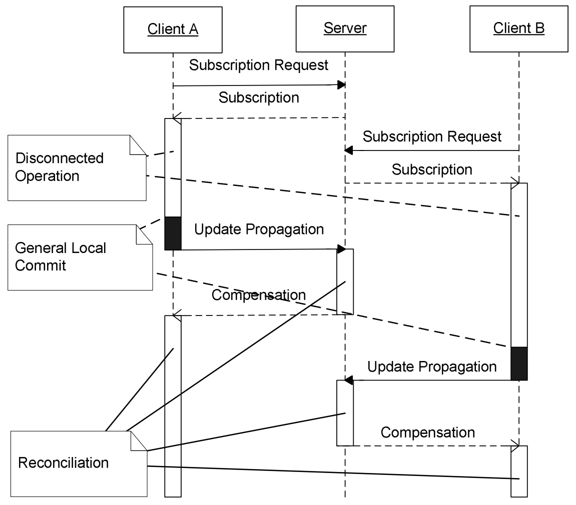

Figure 2. System Lifecycle.

6. Conclusion

The model of the constructed nomadic middleware system supports temporal consistency of data structures semantically associated to XML-documents.

The proposed temporal data model allows efficient management of replicated data in the distributed system. The specially defined high-level operations commute with each other in most cases, thus reducing the number of transaction aborts and increasing system availability. The provided protocol for data reconciliation allows achieving temporal consistency on both client and server sides by processing history replaying according to the algorithm of setting timestamps. To increase the level of transaction concurrency at the server side the technique of data reconciliation may be successfully combined with the concept of accumulators being introduced in [2].

Our future research is focused on extending the model with querying. Also we'll focus on advanced transactions in terms of CoAct [19]. Finally, we plan to improve efficiency of replica management using the concept of accumulators [2] and explore the issues related to cache management.

Acknowledgements. We thank Dr. Boris Novikov from University of Saint-Petersburg, our colleagues Katja Perminova and Oleg Proskurnin from SmartPhoneLabs LLC, Anna Kozlova from JetBrains Inc. and anonymous reviewers of MDDS'05 for their comments which helped us to improve our work.

References

[1] Imielinski, T., Badrinath, B. R. "Mobile wireless computing: Challenges in data management", *Communications of the ACM, Vol. 37, No. 10*, ACM Press, NY, USA, 1994, pp. 19-28.

[2] Kozlova, A., Kochnev, D., Novikov, B. "The Middleware Support for Consistency in Distributed Mobile Applications", *Baltic DB&IS'2004*, Scientific Papers University of Latvia, Vol. 672, 2004, pp. 145-160.

[3] Satyanarayanan, M.: "Fundamental Challenges in Mobile Computing", *15th ACM Symposium on Principles of Distributed Computing'96*, PA, USA, 1996, pp. 1-7.

[4] Kochnev, D.S., Terekhov, A.A. "Surviving Java for Mobiles", *IEEE Pervasive Computing, Vol. 2, No. 2*, 2003, pp. 90-95.

[5] Vossen, G., Weikum, G. Transactional Systems Morgan Kaufmann Publishers Inc, CA, USA, 2001.

[6] Weikum, G., Schek, H.-J. "Concepts and Applications of Multilevel Transactions and Open Nested Transactions", *Elmagarmid, A.K. (ed.) Database Transaction Models for Advanced Applications*. Morgan Kaufmann Publishers Inc, CA, USA, 1992.

[7] Bernstein, P.A., Hadzilacos, V., Goodman, N. *Concurrency Control and Recovery in Database Systems,* Addison-Wesley, Reading, Mass, 1987.

[8] Barghouti, N.S., Kaiser, G.E., "Concurrency control in advanced database applications", *ACM CSUR, Vol 23, No. 3.* ACM Press, NY, USA, 1991, pp. 269-317.

[9] Novikov, B., Proskurnin, O. "Towards collaborative video authoring", *ADBIS'2003*, Dresden, Germany, 2003, pp. 370-384.

[10]Tansel A. U., Clifford J., Gadia S., Jajodia S., Segev A., Snodgrass R. T. (ed.) *Temporal Databases: theory, design, and implementation*, Benjamin/Cummings, 1993.

[11] Etzion O., Jajoda S., Sripada S. (ed.) "Temporal Databases: Research and Practice", *LNCS vol. 1399*, Springer-Verlag, Berlin, 1998.

[12] Goralwalla, I.A., Tansel, A.U., Özsu, M.T. "Experimenting with temporal relational databases", *Conference on Information and Knowledge Management,* Baltymore, Maryland, USA, ACM Press, NY, USA, 1995, pp. 296-303.

[13] Gregersen, H., Jensen, C.S. "Temporal Entity-Relationship Models-A Survey", *IEEE Transactions on Knowledge and Data Engineering. Vol. 11*, 1999, pp. 464-497.

[14] Nørvåg, K. "Temporal Query Operators in XML Database" *ACM SAC'2002,* Madrid, Spain. ACM Press, NY, USA, 2002, pp. 402-406.

[15] Nørvåg, K. "Algorithms for Temporal Query Operators in XML Databases" *XMLDM, Prague, Czech Republic, LNCS, Vol. 2490,* Springer-Verlag, Berlin, 2003, pp. 169-183.

[16] Wang, F., Zaniolo, C. "XBiT: An XML-based Bitemporal Data Model", *23rd International Conference on Conceptual Modeling*, 2004.

[17] Jensen, C.S., Lomet, D.B. "Transaction Timestamping in (Temporal) Databases", *VLDB,* Morgan Kaufmann, 2001, pp. 441-450.

[18] Stavrakas, Y., Gergatsoulis, M., Rondogiannis, P. "Multidimensional XML", *DCW'2000, Quebec City, Canada, LNCS, Vol 1830*, Springer-Verlag, Berlin, 2000, pp. 100-109.

[19] Rusinkiewicz, M., Klas, W., Tesch, T., Wasch, J., Muth, P. "Towards a Cooperative Transaction Model -- The Cooperative Activity Model", *VLDB*, Morgan Kaufmann Publishers Inc, CA, USA, 1995, pp. 194–205.

[20] Proskurnin, O. "Concurent Video: Operational Extensions" *J. Barzdins and A. Caplinskas (ed.), Databases and Information Systems, Selected Papers from the Sixth International Baltic Conference DB&IS'2004,* IOS Press, 2005, pp. 78-87.