Looking for Consistency of Semi-Structured Data*

Ekaterina Gorshkova, Igor Nekrestyanov, Boris Novikov, Ekaterina Pavlova
University of St.-Petersburg, Russia
e-mail:{cathy,igor,katya}@meta.math.spbu.ru, borisnov@acm.org

Abstract

Semistructured data models are becoming an important tool for access to heterogeneous information resources distributed over network. However, consistency issues for semistructured data are not covered yet.

In this paper different approaches to address consistency for semistructured data model are identified. We examine existing consistency models to find out which of them may be appropriate, including incomplete and temporal consistency. Further, we define a model in the frame of multi-level transactions, which relies on abstract high-level operation semantics rather than read-write semantics for definition of concurrency. We then analyze commutativity relationships between basic operations on semistructured data in both data-dependent and data-independent ways.

1 Introduction

Traditional database systems force all data to adhere to an explicitly specified, rigid schema. For many new database applications there can be two significant drawbacks to this approach:

1. The data may be irregular and thus not conform to a rigid schema. In relational systems, null values typically are used when data is irregular, a well-known headache. While complex types and inheritance in object-oriented databases clearly enable more flexibility, it can still be difficult to design an appropriate object-oriented schema to accommodate irregular data.

2. It may be difficult to decide in advance on a single, correct schema. The structure of the data may evolve rapidly, data elements may change types, or data not conforming to the previous structure may be added. These characteristics result in frequent schema modifications, another well-known headache in traditional database systems.

*This work was partially supported by INTAS under grant OPEN-97-11109, Russian Foundation for Basic Research (grant 98-01-00436), and Urbansoft Ltd.

First Russian National Conference on
DIGITAL LIBRARIES:
ADVANCED METHODS AND TECHNOLOGIES,
DIGITAL COLLECTIONS
October 19 - 21, 1999, Saint-Petersburg, Russia

It may be desirable to have an extremely flexible format for data exchange between disparate databases. And even dealing with structured data it may be useful to view it as semistructured for the purposes of browsing.

The semi-structured data model [2] is very suitable when data may be extracted from several independent heterogeneous sources, for example, integrated representations of data exhibited by several WWW sites.

In the context of digital libraries, semistructured data representation may add significant value to more common free-text search technologies. For example, bibliographic data have clearly identified semi-structured nature.

Because of these limitations, many applications involving semistructured data [2] are forgoing the use of a database management system, despite the fact that many strengths of a DBMS (ad-hoc queries, efficient access, concurrency control, crash recovery, security, etc.) would be very useful to those applications.

Currently, the research on semistructured data is concentrated on the following:

1. Data models suitable for representation of semistructured data.

2. Interfaces, especially Web-based, to access semistructured data.

3. Query languages and query evaluation.

As with structured data, it is often desirable to maintain a history of changes to data, and to query over both the data and the changes. Representing and querying changes in semistructured data is more difficult than in structured data due to the irregularity and lack of schema. In [6] a model for representing changes in semistructured data, DOEM and a language for querying over these changes, Chorel is presented. Implementation strategies for the model and query language is discussed. Also the design and implementation of a "query subscription service" that permits standing queries over changes in semistructured information sources is described.

Several different models for semistructured date were described in the literature [1, 4, 11]. Some of these models allow detailed (although flexible) description of data structure (*heavy-weight* models), while others provide only basic definition features and assume dynamic data description techniques (*light-weight* models).

Lore (for Lightweight Object Repository) is a DBMS designed specifically for managing semistructured information. Implementing Lore has required rethinking all aspects of a

DBMS, including storage management, indexing, query processing and optimization, and user interfaces. In [10] an overview of these aspects of the Lore system is provided, as well as other novel features such as dynamic structural summaries and seamless access to data from external sources. Lorel, for Lore Language, is an extension of OQL [5, 3] that introduces extensive type coercion and powerful path expressions for effectively querying semistructured data.

Current research on semistructured data does not examine the dynamic aspects extensively. The following issues may be identified as gaps:

- Extremely flexible but also extremely terse data model should be enriched with more powerful modeling features.
- Consistency issues are not studied at all. To fill this gap, the appropriate framework should be defined, and consistency management techniques should be developed.
- In particular, temporal issues may be considered as essential extension of semistructured data model.
- When the performance is an issue, more efficient storage and indexing structures may be developed and evaluated.

The reason is that in heterogeneous environment consisting of highly autonomous systems one hardly can expect that traditional concepts of database consistency may be applicable.

In this paper we are looking for approaches that may lead to development of the consistency framework for semistructured data. Using of methods automatic information renewing involves necessary support of their consistency. Main goal is to identify the notions related to consistency applicable in heterogeneous distributed decentralized information systems with semistructured data.

Our major focus is on lightweight semistructured data models because they provide more flexible environment which features additional consistency issues.

Traditional consistency frameworks are nor adequate in context where semistructured data may appear. For example, used in relational DBMS consistency constraints are associated with tables, that constitute the data structure. But semistructured data has no clear structure. Changing of notion consistency requires to develop techniques that can ensure consistency of semistructured data in distributed systems. This is especially important in light-weight semistructured data models.

We use light-weight Object Exchange Model (OEM) — the nested self-describing data model designed at Stanford University. The data in this model is represented as a directed labeled graph. Each vertex of such graph is an element of data and each edge has a label. Leaves of this graph are atomic objects which has the type and value. Non-leaf objects, named complex objects, serve as a container for other objects. This model was designed especially for Lore — a DBMS for semistructured data. The queries are expressed in Lorel — the query language used in Lore that is an extension of OQL.

We then identify different approaches to address consistency For semistructured data model. First, we examine different consistency models to find out which existing models may be appropriate.

We define our consistency model in the frame of multilevel transaction model [16]. The consistency is ensured via a notion of serializability, which is based on commutativity relationships between operations of different transactions.

We then identify basic operations on semi-structured data which may be used as a basis for data manipulation algebra, and investigate commutativity conditions for these basic operations.

The paper is organized as follows. In the next section we discuss relevant approaches which may be more suitable for loosely connected autonomous data sources, typically found on the Internet. We then briefly describe relevant features of multilevel transactions. Further, we provide definition of predicates to decide if certain basic operations can commute in Lorel semistructured data manipulation language. These predicates are represented in Lorel itself and therefore depend on the contents of the data store, Next section defines commutativity predicates for graph-based semistructured data model which do not depend on the actual data. Finally, we compare these approaches.

2 Relevant approaches

Consistency issues for semistructured data did not attract much attention till now. Techniques from traditional databases does not suit really well for semistructured case due to its specifics.

Traditional databases systems mostly uses *serializability* as a correctness criteria. However it seems that semistructured databases may need some other correctness criteria to be used. In particular due to the irregular and incomplete nature of semistructured data it seems to be reasonable idea to talk about *incomplete* consistency. This includes usage of approximate correctness criteria such as ϵ-serializability [13, 14, 15].

Yet another potentially interesting approach is based on recent work [7] which presents the notion of incomplete query answers. By defining different *matching* semantics it is possible to simplify usage of semistructured data. In particular traditional consistency restrictions (that are probably too strong for semistructured data) may be relaxed and this will result in increased level of concurrency. Such relaxation may be done as by usage of incomplete queries (or updates) or by direct relaxation of consistency criteria.

Most of semistructured databases import data from external sources through mediators. Imported data are periodically updated by re-importing their current state. This process is similar to updates of temporal data in real-time database systems and raises similar consistency problems. Despite of different scale arise problems are basically the same. Each value of data element is valid only during some interval and every set of data elements is also associated with some relative validity interval. Experience from area of real-time databases seems to be applicable to the case of semistructured data. However import of semistructured data has several unique aspects. One of them is that data imported from different sources may overlap and even contradict each over.

However, the focus of this paper is on more traditional, complete and precise consistency. However, consistency models based on traditional read/write paradigm are too restrictive in heterogeneous distributed environment, especially when primary data sources are not accessible directly due to presence of wrappers and mediators.

One of promising approaches to consistency is based on multi-level transaction model [8, 16]. In contrast with traditional transactions, which are built as sequences of read or write operations, multilevel transactions are constructed as a complex control structure.

Each level of a transaction is considered as a sequence of abstract operations, each of which is considered as atomic for this level (that is, either completes or fails and in the latter case has no effect on the data). In addition, it is assumed that there are no interference between these atomic operations even if they are executed concurrently.

In other words, the abstract operations of any level have certain properties of transactions when considered as complex operations at next lower level.

The major advantage of this model is that once consistency is achieved on one of upper levels, there is no need to care about consistency or any subsuming techniques, such as serializability, at lower levels.

To achieve serializability of transactions at any level, semantical properties of abstract operations should be considered. Usually these properties are expressed via commutativity relationship between operations of concurrently executed transactions.

Given a concurrent schedule, an equivalent schedule may be built using transpositions of subsequent operations (belonging to different transactions provided that these operations commute.

For this reason, commutativity relationships between operations are very important for any consistency considerations.

In the following sections, we investigate commutativity for operations on semistructured data.

3 Content-Dependent Commutativity

In this section we present commutativity predicates which depend on the state of the database, providing for finer granularity of concurrency control. For each pair of generic Lorel operations which potentially do not commute, we specify a Lorel query which tests the commutativity of the operations in question.

We consider the following four generic atomic operations:

1. Selecting the set of objects, that satisfy given criteria.
2. Changing atomic value of the object that satisfies given criteria.
3. Adding the edge from one set of object to another, each of them must satisfy given criteria.
4. Removing the edge from one set of object to another, each of them must satisfy given criteria.

These operations may be considered as a basis for all Lorel data manipulation features.

Any select operations always commute, because they do not change anything in the database.

Now we can consider other possible pairs of operations.

3.1 Update of Atomic Value vs. Update of Atomic Value

To verify if an atomic update operator

update L.X = D
   from $A_1 \ldots A_n$ L
   where P_x(L.X)

commutes with another atomic value update operator

update M.Y = C
   from $B_1 \ldots B_m$ M
   where P_y(M.Y)

the following query may be evaluated:

select L.X
   from $A_1 \ldots A_n$ L, $B_1 \ldots B_m$ R
   where P_x(L.X) and P_y(R.Y)
      and (R.Y = L.X) and (D != C)

If this query returns empty result, then operations commute, otherwise X is a set of objects that can cause a conflict. This is illustrated on figure 1.

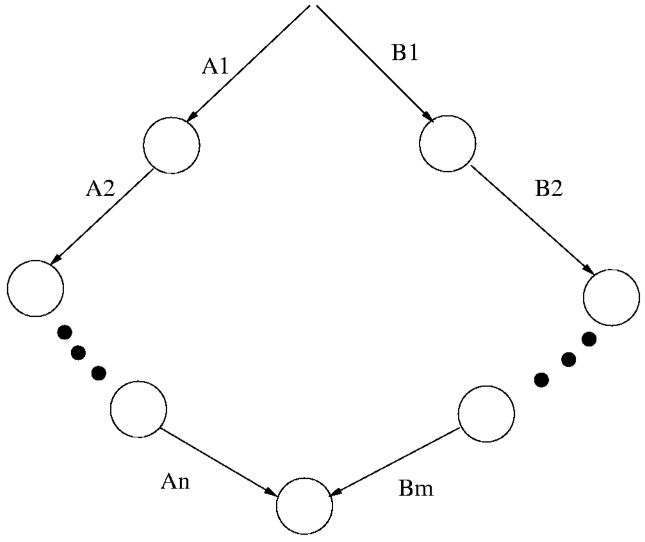

Fig. 1: Two atomic updates

3.2 Select vs. Update of Atomic Value

To verify if a selection operator

select L.X
   from $A_1 \ldots A_n$ L
   where P_x(L.X)

commutes with an atomic value update operator:

update M.Y = D
   from $B_1 \ldots B_m$ M
   where P_y(M.Y)

the following query may be evaluated:

select L.X
   from $A_1 \ldots A_n$ L, $B_1 \ldots B_m$ R
   where P_x(L.X) and P_y(R.Y)
      and (R.Y = L.X) and !P_x(D)

In these queries D is a constant or arithmetic expression (for example $Y \times 3$). If this query returns empty result then operations are commutative, otherwise X is a set of objects that can cause a conflict.

The select and atomic update operators commute if and only if the verifying query returns empty set of objects.

Indeed, suppose the verifying query returns non-empty set of objects — Q. This means, that every member of Q

will be retrieved by the first and second query. If we execute select first, we retrieve all the Q objects. If we execute atomic update first, the objects from Q do not satisfy P_x and so will not be retrieved by select.

Suppose, operations are not commutative. In this case the intersection of the set of objects (Q'), retrieved by select and atomic update queries is not empty. If D satisfies Px, then all the objects updated by atomic update query, will be selected by select query and so operations are not commutative. So, Q' is not empty and $P_x(D)$ returns false. This means that verifying returns not null value.

3.3 Update of Atomic Value vs. Removing an Edge

The conditions for commutation of the third and the fourth operations will be similar. In this section we suppose that paths which appear in the second query do not contain wildcards. For example, we consider removing of the edge and atomic update. In this situation there can be a conflict if an edge that we try to remove in first query appear in the path of the second query. This can happen in two cases as illustrated on figures 2 and 3, respectively.

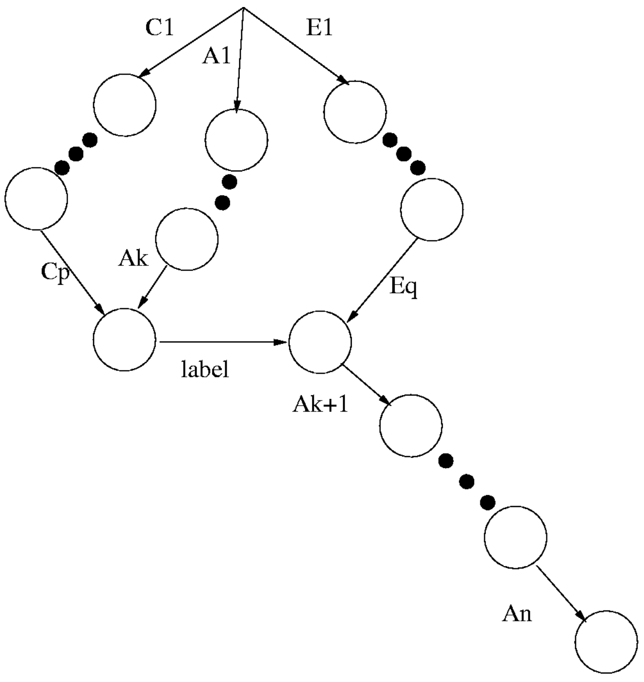

Fig. 2: Edge removal: first case

In first case we have to check commutation between edge removal operator, as shown on 2. The generic edge removal operator may be represented as follows:

```
update Z.label -= W
   from $C_1 \ldots C_p$ Z, $E_1 \ldots E_q$ W
   where $P_{zw}$(Z,W)
```

The atomic value update operator is:

```
update L.X = D
   from $A_1 \ldots A_k.label.A_k + 1 \ldots A_n$ X
   where $P_x$(X)
```

To verify if there can be any conflicts we have to evaluate the query

```
select X
   from $C_1 \ldots C_p$ Z1, $E_1 \ldots E_q$ W1,
      $A_1 \ldots A_k\{Z2\}.label\{W2\}.A_{k+1} \ldots A_n$ X
   where $P_z w$(Z1,W1)
      and (Z1 = Z2)
      and (W1 = W2) and $P_x$(X)
      and not exists V in
         (select $A_1 \ldots A_k\{Z3\}.label\{W3\}.A_{k+1} \ldots A_n$ V
            where $P_x$(V)) : (Z3 != Z1) and (W3 != W1)
```

The atomic update and edge removing operators are commutative if and only if the verifying query returns empty set.

Suppose, the verifying query returns non-empty set of objects — Q. If we execute atomic update query and then execute edge removing query all members of Q will be updated. If we execute edge removal query first then objects in Q will be unreachable for the update query, because the edge "label" will be removed and there is no other path to Q - members which satisfy update query. In this case objects in Q will not be updated and so operations are not commutative.

Suppose, operations are not commutative. In this case the intersection of the set of objects X', retreived by query 1 and query 2 is not empty. If there is any path to object, retrieved by atomic update query, that doesn't share any objects with paths in edge removal query, then operations are commutative, because atomic update will happen independently of removing edge. So, we have that such path cannot exist and veryfying query returns empty set.

In the second case is illustrated in the figure 3.

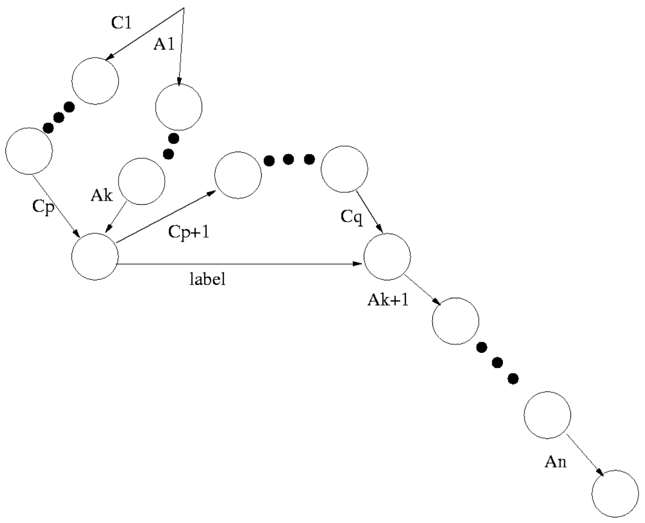

Fig. 3: Edge removal: second case

We consider the same edge removal operator as in the first case and seconde edge removal operator shown below:

```
update Z.label -= W
   from $C_1 \ldots C_p\{Z\}.C_{p+1} \ldots C_q$ W
   where $P_{zw}$(Z,W)
```

The query to check commutation may look like follows:

```
select X
   from $C_1 \dots C_p\{Z1\}.C_{p+1}\{W1\} \dots C_q$ W1
      $A_1 \dots A_k\{Z2\}.label\{W2\}.A_{k+1} \dots A_n$ X
   where $P_z w$(Z1,W1)
      and (Z1 = Z2)
      and (W1 = W2) and $P_x$(X)
      and not exists V in
         (select $A_1 \dots A_k\{Z3\}.label\{W3\}.A_{k+1} \dots A_n$ V
            where $P_x$(V)) : (Z3 != Z1) and (W3 != W1)
```

If this query returns empty result then operations commute, otherwise X is a set of objects that can cause a conflict. This can be motivated in a way similar to outlined above.

3.4 Adding an Edge vs. Adding an Edge

The operations of adding (removing) an edge do not commute when the edge that we are adding (removing) in one query appear in the path expression in the other query. as shown in figure 4.

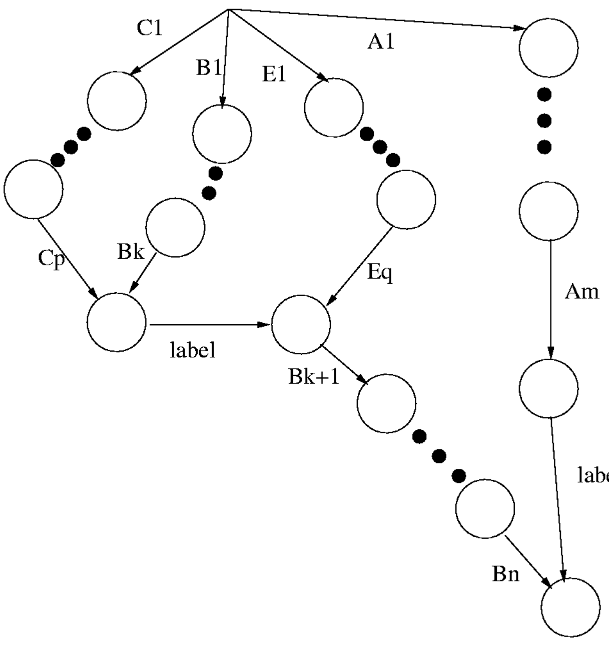

Fig. 4: Two edge insertions

Again, we consider generic Lorel operators. To verify if an edge insertion operator

```
update Z.label += W
   from $C_1 \dots C_p$ Z, $E_1 \dots E_q$ W
   where $P_{zw}$(Z,W)
```

commutes with

```
update X.label2 += Y
   from $A_1 \dots A_m X, B_1 \dots B_k.label.B_{k+1} \dots B_n$ Y
   where $P_{xy}$(X,Y)

select V
   from $C_1 \dots C_p$ Z1, $E_1 \dots E_q$ W1,
      $B_1 \dots B_k\{Z2\}.label\{W2\}.B_{k+1} \dots B_n$ Y
      $A_1 \dots A_m$ X
   where $P_{zw}$(Z1,W1)
      and (Z1 = Z2)
      and (W1 = W2) and $P_{xy}$(X,Y)
```

This can also be proved in a similar way, because to execute query 2, we have to retrieve object from

$$B_1 \dots B_k.label.B_{k+1} \dots B_n$$

like in select or atomic update.

4 Content-Independent Commutativity

In order to consider two given operations as *"commutative"* we need to be sure that these operations do not affect each over. Above approach relies on usage of information about current state of database and only applicable at runtime stage.

In this section we try to overcome this restriction and develop some simple content-independent predicates which guaranties *"commutativity"* of conformed operations.

4.1 Data Model

We use a simple data model which is based on model proposed in [7]. The model is basically a simplified version of OEM [12], Both data and queries are represented by labeled directed graphs. The nodes in a database graph are either complex or atomic. Complex nodes have outgoing edges, while atomic does not. Atomic nodes has values. An atomic value has a type. In every database, there is one distinguished complex node, the root. It is the entry point for accessing the database. Therefore, every node in the database must be reachable from it.

Most of existing languages for semistructured data support navigational queries with regular path expressions. In particular, this applies to a wide range of graph structures in a database and are therefore not restricted to one with prespecified schema. Considered model does not explicitly considers regular path expressions. However some recent works [9] provides the basis for eliminating regular path expressions at compile-time.

In order to specify commutativity predicates we also assume that terminal nodes are not shared, i.e. for each terminal node there is only one edge leading to it.

Formally a semistructured database in this model consists of a rooted finitely branched labeled directed graph $G = (O, r_D, \cdot^D)$ over set of objects O and a function α that maps each terminal node to an atom. r_D denotes *root* node and $\cdot^D$ associates to each label l a binary relation $l^G \subseteq N \times N$ between the nodes (N is a number of objects in O).

4.1.1 Queries

In an abstract view, our queries consists of a set of variables and constraints on the variables. Constraints are divided into *search constraints* and *filter constraints*. As it was suggested in [7] the queries have three components, which resemble whose from SQL-query[1]. The first component, the set of search constraints, is analogous o the FROM-clause. The search constraints forms a labeled directed graph whose nodes are variables: they are a pattern that has to be matched by a part of the database. The second component is a set of filtering constraints, which is analogous to the WHERE-clause. Those matchings which pass the filtering constraints are called *solutions*. The third component is a tuple of *output variables*, which, like SELECT-clause, specifies *answers* as restrictions of the solutions to some variables.

[1]Note that query in Lorel has same components as SQL query.

More formally query is a triple $Q = (G_q, F_q, \overline{x_q})$, where $G_q = (V, r_{D_q}, \cdot^{D_q})$ is a labeled directed graph, called *query graph* those nodes are variables; F_q is a set of filter constraints; and $\overline{x_q}$ is a tuple of variables occurring in V called *output* variables.

4.1.2 Updates

We are extending the model proposed in [7] with formalization of update operations, resembling components from Lorel update operation. In an abstract view update operation consists of *constraints on variables* and *update expression. Constraints on variables* are the same as for query case, namely *search* and *filter* constraints. Unlike SQL-update Lorel update may result not only in changes of values of variables but also may cause addition/removal of some edges[2]. Therefore *update expression* may be specified by set of updating variables, set of edges to add and set of edges to remove.

More formally update operation is a set of five elements $U = (G_u, F_u, \overline{x}_u, \cdot^{D^r}, \cdot^{D^a})$, where G_u and F_u have the same meaning as for query; $\overline{x}_u$ is a set of variables called *updating* variables; $\cdot^{D^r}$ and $\cdot^{D^a}$ are operators which describe set of edges to add and to remove.

4.2 Commutativity predicates

Before presenting our commutativity predicates let introduce used notations. $Lab(\cdot^D)$ denotes the set of labels which $\cdot^D$ maps to non-empty binary relations, i.e. $Lab(\cdot^D) = \{l : \cdot^D(l) \neq \emptyset\}$. We also will use projection $\cdot^D|_{V'}$ of $\cdot^D$ to subset V' of variables V, formally $\cdot^D|_{V'}$ maps each label l to $\{(v_1, v_2) : (v_1, v_2) \in \cdot^D(l)\ ,\ v_1 \in V,\ v_2 \in V'\}$.

In the framework of the above data model we consider only two operation — query and update. Therefore there are only three potential combinations to consider:

- **When do two queries commute?**

  This is the trivial case. No conflicts possible between two queries because they are read-only operations. So, queries are always "commutative".

- **When do query and update commute?**

  In the above notation following condition guaranty commutativity of query and update in framework of our model:

  $$Lab(\cdot^{D_q}) \bigcap \{Lab(\cdot^{D_u}|_{\overline{x}_u} \cup Lab(\cdot^{D^r}) \cup Lab(\cdot^{D^a})\}) = \emptyset$$

- **When do two updates commute?**

  Two update operations $U_1 = (G_{u_1}, F_{u_1}, \overline{x}_{u_1}, \cdot^{D_1^r}, \cdot^{D_1^a})$ and $U_2 = (G_{u_2}, F_{u_2}, \overline{x}_{u_2}, \cdot^{D_2^r}, \cdot^{D_2^a})$ are guaranteed to commutate if following conditions are hold:

  $$Lab(\cdot^{D_{u_1}}) \bigcap \{Lab(\cdot^{D_{u_2}}|_{\overline{x}_{u_2}} \cup Lab(\cdot^{D_2^r}) \cup Lab(\cdot^{D_2^a})\}) = \emptyset$$

  and

  $$Lab(\cdot^{D_{u_2}}) \bigcap \{Lab(\cdot^{D_{u_1}}|_{\overline{x}_{u_1}} \cup Lab(\cdot^{D_1^r}) \cup Lab(\cdot^{D_1^a})\}) = \emptyset$$

[2] Addition/removal of edges may cause change of overall data structure that is natural in the context of semistructured data but does not suit well with relational model.

Note that operations in question may actually commute even if corresponding predicate is not hold.

These predicates are pretty rough and probably may be significantly improved. However they do not require usage of any runtime information and, therefore, may be applied during the compilation stage. This makes them potentially very useful for preliminary planning and optimization.

5 Conclusion

In this paper we are investigating consistency issues related to semistructured environments. Major focus of this paper is specification of commutativity relationships in the frame of multi-level transactions. The commutativity is essential for establishing consistency when high-level abstract operations are considered instead of read/write basic operations.

The commutativity predicates are defined in two ways: as queries in Lorel language and as predicates over graph model of semistructured data.

The advantage of the former method is that predicates expresses as queries depend on the contents of the database and therefore provide more precise commutativity criteria, potentially enabling better degree of concurrency. The advantage of the latter method is that predicates depend only on operators and therefore can be evaluated in advance, potentially providing better performance.

Although major focus of the paper may result in a precise consistency considerations, we expect that approximate approaches, such as relative consistency, may be more promising. We are planning to investigate these approaches in our future work.

References

[1] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. Wiener. The lorel query language for semistructured data. *Journal of Digital Libraries*, 1(1), November 1996.

[2] S. Abiteboul. Querying semistructured data. In *Proceedings of the International Conference on Database Theory*, Delphi, Greece, January 1997.

[3] F. Bancilhon, C. Delobel, and P. Kanellakis. *Building an Object-Oriented Database System: The Story of O2.* Morgan Kaufmann, San Francisco, California, 1992.

[4] P. Buneman, S. Davidson, G. Hillebrand, and D. Suciu. A query language and optimization techniques for unstructured data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Montr'eal, Qu'ebec, June 1996.

[5] R. Cattell. *The Object Database Standard: ODMG-93.* Morgan Kaufmann, San Francisco, California, 1994.

[6] Sudarshan S. Chawathe, Serge Abiteboul, and Jennifer Widom. Representing and querying changes in semistructured data. In *Proceedings of the Fourteenth International Conference on Data Engineering*, pages 4–13, Orlando, Florida, USA, February 1998.

[7] Yaron Kanza, Werner Nutt, and Yehoshua Sagiv. Incomplete answers for queries over semistructured data. In *Proceedings of the 6th International Workshop on Knowledge Representation meets Databases (KRDB'99)*, January 1999.

[8] D. Lomet. MLR: A recovery method for multi-level systems. Technical Report CRL 91/7, Digital Equipment Corporation, Cambridge Research Lab, July 1991.

[9] Jason McHugh and Jennifer Widom. Compile-time path expansion in Lore. In *Proceedings of the Workshop on Query Processing for Semistructured Data and Non-Standard Data Formats*, Jerusalem, Israel, January 1999.

[10] J. McHugh, S. Abiteboul, R. Goldman, D. Quass, and J. Widom. Lore: A database management system for semistructured data. Technical report, Database Group, Stanford University, February 1997.

[11] G. Mecca, P. Atzeni, A. Masci, P. Merialdo, and G. Sindoni. From databases to web-bases: The araneus experience. Technical Report 34-1998, Universita' di Roma Tre, Dipartimento di Informatica e Automazione,, May 1998.

[12] Y. Papakonstantinou, H. Garcia-Molina, and J. Widom. Object exchange across heterogeneous information sources. In *Proceedings of the Eleventh International Conference on Data Engineering*, pages 251 - 260, Taipei, Taiwan, March 1995.

[13] Calton Pu and Avraham Leff. Autonomous transaction execution with epsilon-serializability. In *Proceedings of 1992 RIDE Workshop on Transaction and Query Processing, IEEE/Computer Society*, Phoenix, February 1991.

[14] Calton Pu. Generalized transaction processing with epsilon-serializability. In *Proceedings of Fourth International Workshop on High Performance Transaction Systems*, Asilomar, California, September 1991.

[15] Krithi Ramamritham and Calton Pu. A formal characterization of epsilon serializability. *IEEE Transactions on Knowledge and Data Engineering*, December 1995.

[16] Gerhard Weikum. Principles and realization strategies of multilevel transaction management. *ACM Transactionson Database Systems*, 16(1):132–180, March 1991.