***S. V. COOX,*** SPb State University postgraduate student, (St. Petersburg)
***A. A. SIMANOVSKY,*** SPb State University postgraduate student, (St. Petersburg)

## REGULAR EXPRESSIONS IN XML SCHEMA EVOLUTION

A data schema is a set of constructs that emulate relations between domain instances. Schema evolution is changes in the data schema and a system of data consistency support for such modifications. Axiomatic model regulates modifications and maintains data consistency by ensuring that the data schema always satisfies a set of axioms.

This paper is an extension to previous work, which has defined an axiomatic model for XML-database schema evolution. A DTD regular expressions mechanism is incorporated into the axiomatic model. A new classification of modifications allowable in the model is presented.

**1. Introduction** XML as a data storage becomes more and more popular in recent years. It is used not only as a Web interface to traditional databases but as standalone data storage as well. In both cases it is appropriate to talk about XML database with structure defined by a collection of Document Type Definitions (DTDs). XML data is stored in a collection of valid XML documents [4] with structure regulated by corresponding DTDs.

As any data storage an XML database may evolve as the time passes. The problem of schema evolution also stands for XML in the task of merging similar documents to a single database. In the case a sequence of simple schema changes (called schema transformation) is to be built. An axiomatic model developed in the paper is purposed to support XML schema evolution and transformation. The model presented extends the model described in [1] with support of regular expressions.

The remainder of the paper is organized as follows. The basic data model is described in Section 2. Section 3 extends the model to support regular expressions in DTD. Section 4 provides an axiomatization of the extended model. In Section 5 we classify schema evolution changes and express the changes in terms of the axiomatic model. An overview of related work is presented in Section 6. Section 7 summarizes the work.

**2. Basic data model.** The data contained in an XML-document is stored in tags with their attributes and their nesting model. The model presented in [1] utilizes this observation and represents XML data as a set of tags with their attributes and the tags hierarchy (By the term hierarchy we mean parent-child relations between tags representing tag nesting not regarding the conditions of their nesting.). This Section contains a brief description of the model explained in [1] (further referred as the *basic model*).

The basic model requires that input DTDs have no recursion (including indirect) in tags definitions. For example, the construct

```
<!ELEMENT foo (bar*,...)>
<!ELEMENT bar (foo*,...)>
```

is allowed by XML syntax, but not allowed by the model. Note that real-life DTDs rarely use recursion.

```
<!ELEMENT Message (From,To,Subject,Body>
<!ELEMENT From (#PCDATA)>
<!ELEMENT To (#PCDATA)>
<!ELEMENT Subject (#PCDATA)>
<!ELEMENT Body (Greeting,P+|(Quote,P+)*,Fairwell)>
<!ELEMENT Quote (P+)>
  <!ATTLIST Quote name CDATA #REQUIRED>
<!ELEMENT P (#PCDATA)>
<!ELEMENT Greeting (#PCDATA)>
<!ELEMENT Fairwell (#PCDATA)>
```

**Fig. 1.** Working example of DTD.

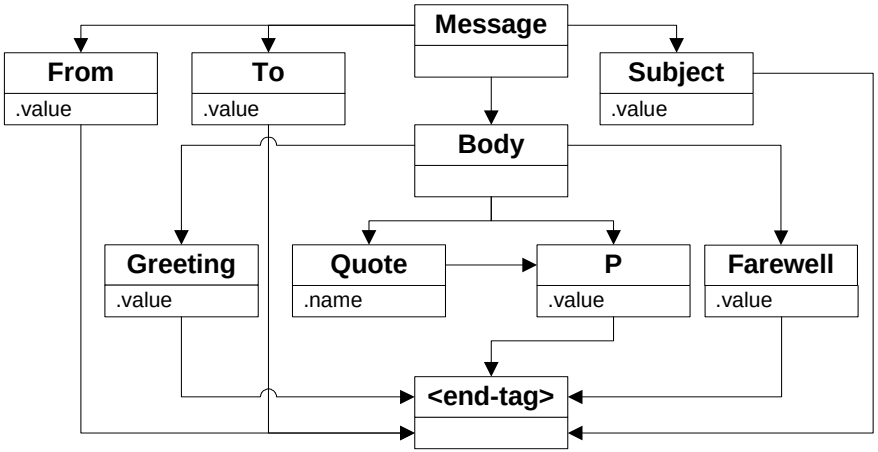


**Fig. 2.** Basic XML schema.

For each DTD in a database an acyclic directed graph with exactly one root vertex is built. Vertices of the graph are tags and edges are tag inclusions. Each vertex receives a correspondent set of tag attributes, which are the data stored in the document (simple tags are considered as tags with attribute of #PCDATA type). We add to the graph a dummy vertex <end-tag> included into all leaf-tags. Fig.s 1 (a DTD used throughout the paper as a working example) and 2 (a basic schema for the DTD) illustrate the process. We consider a tag attributes together with attributes

of enclosing tags. For example, the set of attributes of P will be {Quote.name, P.value}.

Typical schema changes in schema evolution are addition and deletion of types (tags) and their properties (attributes) and modifications in hierarchy. A highly automated support of database reorganization is the main purpose of an axiomatic model of the schema. The model being described uses notation and axiomatic model of the TIGUKAT system developed to support object-oriented database schema evolution [6, 5].

| *Term* | *Description* |
|---|---|
| $\mathcal{T}$ | Tag hierarchy |
| s, t, T, $\perp$ | Tags |
| P(t) | A set of immediate predecessors of t. |
| $P_e(t)$ | A set of essential predecessors of t |
| PL(t) | A hierarchy of all predecessors of t |
| N(t) | A set of "native" attributes of tag t |
| H(t) | A set of "inherited" attributes of tag t |
| $N_e(t)$ | A set of essential attributes of tag t |
| I(t) | A set of all attributes of tag t (interface) |
| $\alpha_x(f,\mathcal{T}')$ | Apply-all operation |

**Table 1.** Notation used in the axiomatic system

$\mathcal{T}$ – A set of all tags in the system and a graph of the system. $\mathcal{T}$ of the working sample is shown on Fig. 2.

The hierarchy $\mathcal{T}$ is defined by sets of *immediate predecessors P(t)* for each tag t. Immediate predecessors are tags that explicitly include tag t and do not include t indirectly (through a transitive inclusion). In Fig. 2 Body is included into P explicitly and through Quote, so P(P) = {Quote}.

*Essential predecessors $P_e(t)$* are tags specified by a schema designer as essential for the tag t. A tag s of $P_e(t)$ should be maintained as a predecessor of t as long as possible until explicitly removed from $P_e(t)$, or from the system. The restriction for the set $P_e$ is $P(t) \subseteq P_e(t)$, that is, all immediate predecessors are essential. If we specify $P_e(P)$ = {Quote, Body} (Fig. 2) and tag Quote is removed, then Body becomes immediate predecessor of P, since it is essential for it.

*A hierarchy of all predecessors PL(t)* of a tag t is a sub-graph of $\mathcal{T}$, including all the tags, from which t can be reached, and t. Obviously, PL(t) has all the properties of original graph: it is acyclic, it has exactly one root (T) and the only last vertex (t). For example, PL(P) = { Quote, Body, Message }

*Native attributes N(t) of a tag t* are the attributes defined in it. For example, N(Farewell) = {value}.

*Inherited attributes H(t) of tag t* are the union of attributes of all predecessors of t. H(P) = {Quote.name }. The set contains all the attributes that may appear in an XML document on the route from the root to the tag P. Sets of native and inherited attributes of a tag are disjoint.

*Essential attributes $N_e(t)$ of tag t* are the attributes declared by a schema designer as essential for t. The set contains all native and some of inherited attributes ($N(t) \subseteq N_e(t)$). If an attribute is included into $N_e(t)$, it will remain attribute of t (native or inherited) as long as possible. For example, if $N_e(P)$ = {value, Quote.name} and Quote is removed from the system, then P gets new attribute QuoteName.

An *Interface I(t) of tag t* is the union of all its attributes ($N(t) \cup H(t)$).

In axiomatic model we write apply operations as $\alpha_x(f,\mathcal{T}')$. It denotes a union of sets returned by application of unary operation f to all elements of set $\mathcal{T}' \subseteq \mathcal{T}$. The function f is applied to a variable *x*, denoted as a subscript of $\alpha$. If $\mathcal{T}'$ is empty, the result is also empty.

Table 2 describes all the sets in terms of $P_e(t)$ and $N_e(t)$. All the schema changes can also be expressed in these terms. The sets $P_e(t)$ and $N_e(t)$ are defined by schema designer and is changed to reflect schema modifications.

| | |
|---|---|
| $\forall t \in \mathcal{T},\ P_e(t) \subseteq \mathcal{T}$ | (1) |
| $\forall t \in \mathcal{T},\ t \notin \cup\, \alpha_x(PL(x),P(t))$ | (2) |
| $\exists T \in \mathcal{T}, \forall t \in \mathcal{T} \mid T \in PL(t) \wedge P_e(T) = \varnothing$ | (3) |
| $\exists \perp \in \mathcal{T}, \forall t \in \mathcal{T} \mid t \in PL(\perp)$ | (4) |
| $\forall t \in \mathcal{T},\ P(t) = P_e(t) - \cup\alpha_x(PL(x) \cap P_e(t) - \{x\}, P_e(t))$ | (5) |
| $\forall t \in \mathcal{T},\ PL(t) = \cup\alpha_x(PL(x),P(t)) \cup t$ | (6) |
| $\forall t \in \mathcal{T},\ I(t) = N(t) \cup H(t)$ | (7) |
| $\forall t \in \mathcal{T},\ N(t) = N_e(t) - H(t)$ | (8) |
| $\forall t \in \mathcal{T},\ H(t) = \cup\alpha_x(I(x),P(t))$ | (9) |

**Table 2.** Axiomatic model describes the schema in terms of $P_e$ and $N_e$.

Satisfying the axioms ensures automatic support for schema changes (the only thing to do is proper updates of the $P_e$ and $N_e$ sets). Axioms (1)-(4) define the structure of a graph (acyclicity, existence and uniqueness of the root and end tags), other five axioms define sets P, PL, I, N and H.

(1) *Axiom of Closure* states that all the tags of $\mathcal{T}$ are sub-tags of other tags of $\mathcal{T}$.

(2) *Axiom of Acyclicity* provides the absence of cycles in a graph.

(3) *Axiom of Rootedness* states that there exists exactly one tag T (Message in the example) that contains all tags. The axiom can be relaxed to allow a forest of schemas (DTDs). In the case we can work with each DTD separately to satisfy the axiom.

(4) *Axiom of Pointedness* states that there exists exactly one tag $\perp$, which is contained (explicitly or indirectly) in every other tag of a system. This element added to the schema for convenience (in order not to handle leaf tags in a specific way).

(5) *Axiom of Immediate Predecessors* defines subset of essential predecessors such that there is no way to the current tag via another tag of the system. The axiom allows building immediate predecessors sets if all sets of essential predecessors are known.

(6) *Axiom of Predecessors Graph* defines a subgraph, consisting of tag t and all predecessors of its immediate predecessors. The axiom allows building the set PL(t).

(7) *Axiom of Interface* defines tag interface (all its attributes) as a union of the sets of native and inherited attributes. The axiom allows building interface for all tags.

(8) *Axiom of Nativeness* states that native attributes of a tag are its essential attributes that are not inherited. This allows building of N(t) sets.

(9) *Axiom of Inheritance* states that inherited attributes are the union of interfaces of its predecessors. This axiom allows building of H(t) sets.

We illustrate the axiomatic model using the graph on Fig. 2. Axioms 1, 3 and 4 are satisfied (all the tags are of *T*, T = Message, $\perp$ - added dummy tag <end-tag>). Axiom 2 (acyclicity): there is no tag that precedes its immediate predecessor. Axiom 5 (immediate predecessors): suppose we have the following sets

$P_e(P)$ = { Quote, Body, Message }
$P_e(Quote)$ = { Body, Message }
$P_e(Body)$ = { Message }
Make sure that set of immediate predecessors P(P) = {Quote}:

$X = P_e(P) - \cup\alpha_x(PL(x)\cap P_e(P) - \{x\}, P_e(P)) =$
$\{Quote, Body, Message\} - \cup\alpha_x (PL(x)\cap\{Quote, Body, Message\} - \{x\},\{Quote, Body, Message\}) =$
$\{Quote, Body, Message\} - ((PL(Message)\cap\{Quote, Body, Message\} - \{Message\}) \cup (PL(Body)\cap\{Quote, Body, Message\}-\{Body\})) \cup (PL(Quote)\cap\{Quote, Body, Message\}-\{Quote\}))$
PL(Message) = {Message}
PL(Quote) = {Quote, Body, Message}
PL(Body) = {Body, Message} =>
$X = \{ Quote, Body, Message \} - (\varnothing \cup \{Message\}\cup \{Body, Message\}) =$
{Quote} = P(P)

The axiomatic model provides mechanism for recalculating the schema after modifications in sets of essential predecessors $P_e(t)$ and attributes $N_e(t)$ are done. The changes in these sets are fundamental to schema evolution, and the axiomatic model expresses relations between tags in terms of these sets.

**Theorem 1:** The schema axioms are sound and complete. In other words, if the sets $P_e(t)$ and $N_e(t)$ are defined for each tag, then a correct schema satisfies the axioms and making changes to the schema is governed by the sets.

The complete proof and comments can be found in [1].

**3. Extended data model.** The model described does not match the source DTD exactly, it lacks for representation of inter-tag relations. To address this we extend the schema with regular expression handling. Regular expressions for DTD are described in [4], a general description can be found in [2]. The basic model does not provide any means for reflecting the difference, for example, between the following DTD declarations:

```
<!ELEMENT Body (Greeting, P+|(Quote, P+)*,Farewell)>
```

and

```
<!ELEMENT Body (Greeting, Quote, P, Farewell)>.
```

To distinguish such cases, we sophisticate our data structure model by introducing special nodes representing regular expressions. We call such nodes the s-nodes. The s-nodes represent the structure of a regular expression in the element declaration. Additional axioms regulate the placement of the s-nodes in the graph. We also update the basic transformations classification. Prior to introducing new axioms we describe changes to be introduced into the basic model.

The nodes of the basic model are passed to the extended model. We call such nodes the g-nodes. However, they are connected with each other no more. To represent their relations we add s-nodes. Each s-node represents a sequential placement of its sons or a choice between its sons. For the example the following structure is created to represent the regular expression describing the containment of the Greeting, P and Quote tags in the Body tag.

Edges between two s-nodes are called the s-s edges, between an s-node and a g-node – s-g edges, between a g-node and s-node – g-s edges. There are no g-g edges in the new graph. The nesting relation between two tags is represented by a path of (g-s, s-s,…, s-s, s-g) kind in the new graph. Further we also use the term path for a set of nodes that reside on the path, that is, nodes, for which there is both incoming and outgoing edge contained in the path. For example, on Fig. 4 a path from the node marked Body to the node marked Quote consists of two nodes marked with ‘,’ and one marked with ‘|’.

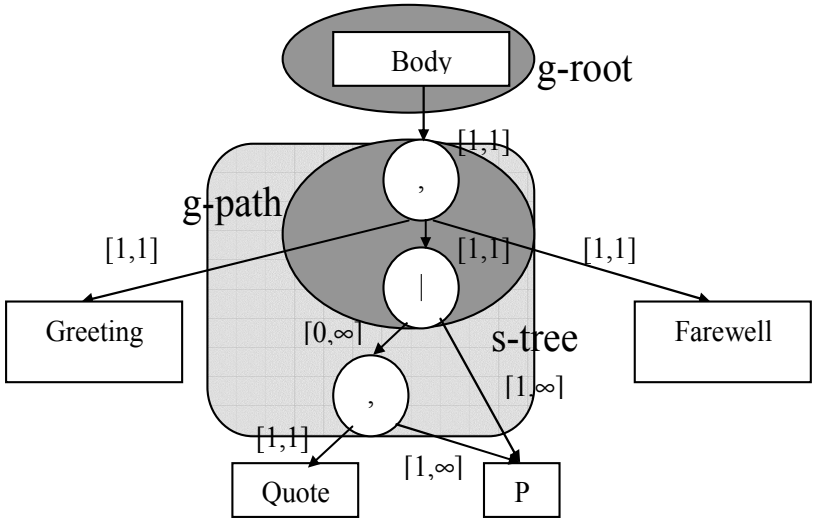


**Fig. 4**. The s-tree of Body tag.

Each edge has a mark that consists of a pair of non-negative integer numbers including infinity sign. A pair represents the relation kind. (0,∞) stands for Kleene star ('*'), (1,∞) – for Kleene plus ('+'), (0,1) – for none-or-one ('?'), and (1,1) – for default (one-to-one) relation.

Each s-node receives either ',' or '|' mark. It is called a **seq**- or an **or**-node respectively. For an **or**-node we require that it has at least two children. Note that s-nodes make up a forest of trees. We call each tree of that forest an s-tree. The graph for our working example is presented on Fig. 5.

Each s-tree is attached to a controlling g-node. We call such node a *tag-owner* (g-root) of that node.

A *containment path* (g-path) is the path from the given s-node to the tag-owner of the s-tree with inclusion of that tag-owner, or a vertex itself if it is a g-node.

An *immediate containers set* (IN set) for g-nodes is a set of new graph vertices, from which there exists an edge to the given node, and there is no path from that node to the given one such that it goes through a g-node (here and below "goes though" means that the path has both, incoming and outgoing, edges for that vertex). That is, the IN(t) set is an analogue to the P(t) set in the basic model. The immediate containers set of an s-node consists of its parent. We require that for any s-node there exists a node that includes the s-node into its immediate containers set. For an inner node of an s-tree such node is one of its children, for an s-tree leaf such node may not exist in original DTD, to fulfill the requirement we add two nodes (a g-node that will contain the given s-node in the IN set, and an s-node, through which the new g-node will be connected to the end tag) per each case like that.

The *essential attributes*, *inherited attributes* and *interface* of a g-node are defined as the corresponding notions in the basic model. For s-nodes all attribute sets are empty.

Besides these new sets we retain the old graph vertices sets of immediate predecessors (P set), essential predecessors ($P_e$ set), and hierarchy of all predecessors (PL set). We extend the notions over s-nodes and define them in terms of the new graph. Thus, *immediate predecessors* (P set) of a g-node t becomes a set of all g-nodes, for which there exists a path to t and all such paths do not go through other g-nodes; *hierarchy of all predecessors* (PL set) of a g-node t becomes a set of g-nodes, from which t is reachable; *essential predecessors* ($P_e$ set) of a g-node t contains the tags specified by schema designer as essential for t and must include P(t). Tags from $P_e$ set should remain predecessors of t as long as possible. For an s-node we define P set as the parent of that node, the $P_e$ and PL sets as a path from that node to the tag-owner of the s-tree.

We also define another set that was not present in the basic model, the set of *essential containers* ($IN_e$ set). The $IN_e$ set of an s-node is the same as its $P_e$ set. The $IN_e$ set of a g-node is defined by the schema designer with the following constraints:

- The $IN_e$ set consists of s-nodes.
- The tag-owners of the elements of the $IN_e$ set compose the $P_e$ set.
- The $IN_e$ set contains the IN set.

At last, we define a reduction operation for the new graph. The reduction is done as follows:

- Marks are removed from the edges.
- The paths of type (g-s, s-s, ..., s-s, s-g) in the graph are replaced with a single edge that starts at the beginning of the path and ends at the end.
- Duplicate edges are replaced with one.

Note, that if we perform a reduction for the extended model all statements concerning the basic model hold true. Indeed, as Theorem 1 states, the basic model axiom set is complete. That is, if the sets P, $P_e$, and PL that we define for the extended model are the same sets that we defined in the basic model they will necessarily comply with the axiom set of the basic model.

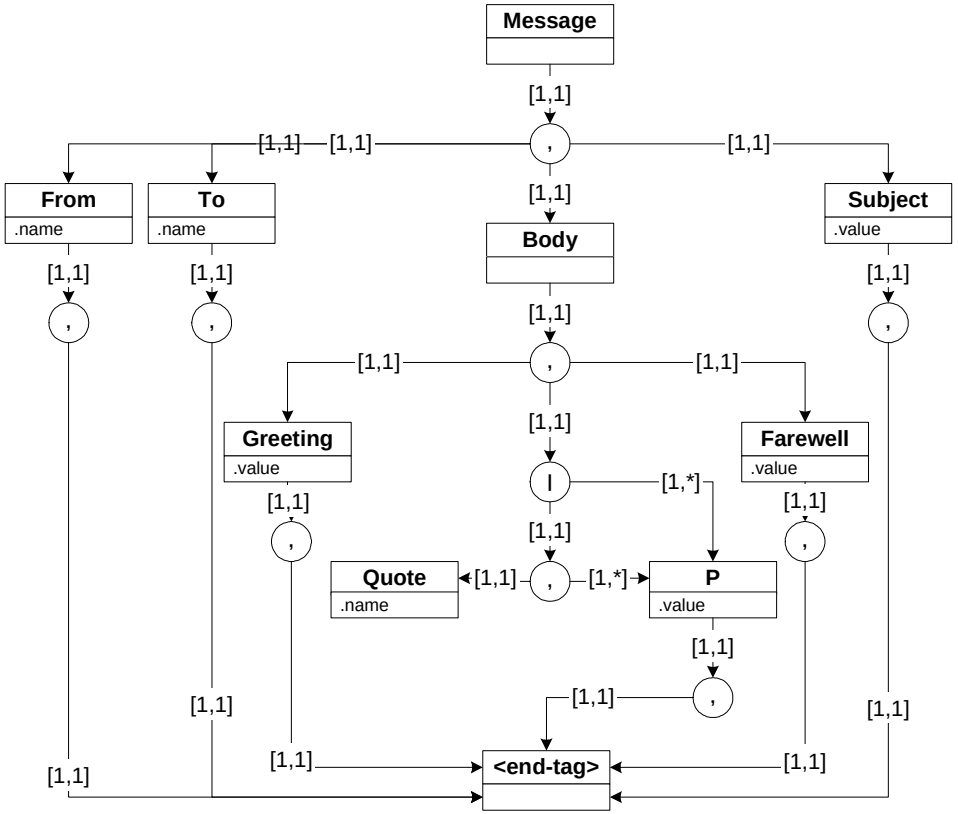


**Fig. 5**. The graph of the extended model.

**4. Schema axiomatization.** We do not distinguish a one-element set and its only element and use them in place of each other. For example, when a function expecting a vertex is applied to a set of vertices, it means that the set consists of one element and the function is applied to it

| | |
|---|---|
| $g(t) = t \vee g(t) = \varnothing;\ s(t) = t \vee s(t) = \varnothing$ | (1.1) |
| $\cup\, \alpha_x(g(x), T) \ \cup\ \cup\alpha_x(s(x), T) = T$ | (1.2) |
| $\cup\, \alpha_x(g(x), T) \ \cap\ \cup\alpha_x(s(x), T) = \varnothing$ | (1.3) |

**Table 3.** Node-type mappings g and s: $g : T \rightarrow T^*$, $s : T \rightarrow T^*$.

Mapping axioms ensure that if g(t)=t then s(t)=∅, and vice versa, and for all vertices either g(t)=t or s(t)=t. In the former situation we call t a g-node and in the latter we call t an s-node. That is, the g and s mappings are the characteristic functions of the disjoint sets of g- and s-nodes.

| | |
|---|---|
| $\forall t \in \cup\alpha_x(s(x), T)\ \ \|IN(t)\| = 1\ \&\ IN(t) \neq t$ | (2.1) |
| $\forall t \in \cup\alpha_x(g(x), T)\ \ IN(t) \subseteq \cup\alpha_x(s(x), T)$ | (2.2) |
| $\forall t \in \cup\alpha_x(s(x), T)\ \ IN(s) \in \cup\alpha_x(g(x), T)\ \Rightarrow \forall s \neq t\ \ IN(s) \cap IN(t) = \varnothing$ | (2.3) |
| $\forall t \in \cup\alpha_x(s(x), T)\ \ \exists s\ \ t \in IN(s)$ | (2.4) |
| $\forall t \in \cup\alpha_x(g(x), T)\ \ G\text{-root}(t) = t;\ \forall t \in \cup\alpha_x(s(x), T)\ G\text{-root}(t) = G\text{-root}(IN(t))$ | (2.5) |
| $\forall t \in \cup\alpha_x(g(x), T)\ \ G\text{-path}(t) = G\text{-root}(t);$ $\forall t \in \cup\alpha_x(s(x), T)\ \ G\text{-path}(t) = G\text{-path}(IN(t)) \cup IN(t)$ | (2.6) |
| Mark of s-node t is '\|' $\Rightarrow \exists u,v\ \ u \neq v\ \ \&\ \ t \in IN(u)\ \&\ t \in IN(v).$ | (2.7) |

**Table 4.** The *s-tree axioms* define the s-tree structure. The Axioms 2.1 (single parent) and 2.2 (axiom of delimiter) define a forest of trees on the s-nodes in the graph. Each s-tree has an owner-tag (2.3). Axiom 2.4 (connectivity) requires each s-node to have children, each **or**-node has at least two children (2.7).

| | |
|---|---|
| $\forall t \in \cup\alpha_x(s(x), T)\ \ P_e(t) = IN_e(t) = PL(t) = G\text{-path}(t),\ P(t) = IN(t)$ | (3.0) |
| $\forall t \in \cup\alpha_x(g(x), T)\ \ \ IN_e(t) \subseteq \alpha_x(s(x),T);\ \ P_e(t) = \cup\alpha_x(G\text{-root}(x), IN_e(t))$ | (3.1) |
| $\forall t \in \alpha_x(g(x),T),\ t \notin \cup\, \alpha_x(PL(x),P(t))$ | (3.2) |
| $\exists T \in \alpha_x(g(x),T),\ \forall t \in T\ \ T \in PL(G\text{-root}(t)) \wedge P_e(T) = \varnothing$ | (3.3) |
| $\exists \perp \in \alpha_x(g(x),T),\ \forall t \in T \mid G\text{-root}(t) \in PL(\perp)$ | (3.4) |
| $\forall t \in \alpha_x(g(x),T),\ P(t) = P_e(t) - \cup\alpha_x(PL(x) \cap P_e(t) - \{x\}, P_e(t))$ | (3.5) |
| $\forall t \in \alpha_x(g(x),T),\ PL(t) = \cup\alpha_x(PL(x),P(t)) \cup t$ | (3.6) |
| $\forall t \in T,\ I(t) = N(t) \cup H(t)$ | (3.7) |
| $\forall t \in T,\ N(t) = N_e(t) - H(t)\ ;\ t \in \alpha_x(s(x),T) \Rightarrow N_e(t) = \varnothing$ | (3.8) |
| $\forall t \in T,\ H(t) = \cup\alpha_x(I(x),P(t))$ | (3.9) |
| $\forall t \in \cup\alpha_x(g(x), T)\ \ IN(t) = \cup\alpha_x(\chi_{G\text{-root}(x) \in P(t)},\ IN_e(t))$ | (3.10) |

**Table 5.** The revision of the axioms of the basic model and a *connection axiom* (3.10) for IN and P sets of a g-node. $\chi_{<condition>}$ is a characteristic function of a set defined by <condition>.

**Lemma 1**. Axioms 3.1-3.9 define a basic model over g-nodes of the model, and the basic model defined by these axioms is the one obtained from the original model by applying reduction operation.

**Proof**. Indeed, from Axiom 3.1 follows the first axiom of the basic model, namely, that the $P_e$ sets are contained in the set of schema tags. Axiom 3.2 together with s-tree axioms implies the acyclicity axiom of the basic model. Axiom 3.3 and s-tree axioms state that there exists a root in the g-nodes hierarchy. Axiom 3.4 together with s-tree axioms guarantees the existence of the terminating vertex. The axioms that express dependencies of P, $P_e$ and PL sets are the same as in the basic model. Axioms 3.7-3.9 for the g-nodes are also the same as in the basic model. Thus, Axioms 3.1-3.9 create a basic model over the g-nodes.

It is left to show that provided that mappings g, s and $IN_e$ sets are the same, Axiom 3.1 and the definition from Section 3 define the same $P_e$ sets. Then it would follow from Theorem 1 that the basic models defined by the axioms and obtained from the reduction are necessarily the same. To show that it is true, it is enough to prove that the axioms define the same s-trees as definitions from Section 3, and this is clearly seen since they both require the same contents of IN, $IN_e$, P, $P_e$, and G-root sets for s-nodes. Thus, the basic models obtained through reduction and defined by Axioms 3.1-3.9 are necessarily the same. •

To describe the edge marks we introduce a mapping M, which takes an edge and returns a pair of integers. We define a product of two pairs of integers as a pair of products of their first and second constitutes of pairs:

(a, b) * (c, d) = (a*c, b*d).

We require the following axiomatic property for operations except changing mark explicitly, which we call the *marking rule*:

$\Pi_{s \in \text{G-path}(t) \cup t} M(s, IN(s)) = \text{const}.$

The total axiom set consists of the mapping axioms, s-tree axioms, basic axioms, and the marking rule. We formulate a theorem analogous to the one formulated for the basic model.

**Theorem 2**: The introduced set of axioms is consistent and complete. We assume consistency and completeness here in the sense that the axioms allow non-contradictory and non-ambiguous building of all sets of the model.

**Proof**. First, we will show that the definitions of sets provided in Section 3 satisfy axioms, which proves consistency. Then, the completeness will be proved by deriving an unambiguous algorithm building all sets defined by the axioms.

The G-root and G-path sets obviously satisfy the axioms. To show that P, $P_e$ and PL sets comply with the model, we apply reduction operation. Since after reduction we are in the conditions of the basic model, we can use Theorem 1. That is, we have a set of axioms that defines the contents of the P, $P_e$ and PL sets, namely, the set of axioms of the basic model. Since, as Lemma 1 states, that set of axioms is included into the axioms of the extended model, they define the same sets. Obviously, the IN set for s-nodes complies with the axioms. Now we need to show that the IN set for g-nodes as defined in Section 3 complies with the s-tree axioms and the connection axiom. Satisfying s-tree axioms is obvious. To show that IN set satisfies connection axiom, we employ results for the basic model once more: we use the definition of the P set as it is given in Section 2 rather than its axiomatic formula (it is possible because after reduction holds Theorem 1). The definitions of P, IN and $IN_e$ sets from Section 3 satisfy the connection axiom, which states that IN set consists of those $IN_e$ set elements, which G-roots belong to the P set. Thus, sets described in Section 3 comply with the extended model axioms; consistency is proven.

It is obvious how to build all sets for s-nodes, as well as G-root and G-path sets for g-nodes from the extended model axioms. To build $P_e$, P and PL sets, we use the same algorithm as in the basic model: it is applicable because Axioms 3.1-3.9 define the basic model, as it was shown in Lemma 1. Axiom 3.10 allows building the IN set. Thus, the axioms allow building sets unambiguously. The completeness is proven. •

In the next Section we show how the axioms regulate schema evolution by modification of sets enabling semi-automatic database schema updates.

**5. Schema evolution.** In this section we classify the basic changes of the schema and formulate them in terms of the axiomatic model. The question of applying these changes to the documents satisfying a given schema, though being of certain interest, is out of the scope of this paper.

We distinguish two general types of changes: changes concerning attributes, that is, the changes of the data itself, and changes concerning data structure, that is, the changes of tag relations. The first type of changes requires modification of $N_e$ set, the second type – modification of $IN_e$ sets. The operations of the first type are the same as in the basic model.

1. **Adding an attribute to a tag t**. The attribute is included into $N_e(t)$. The I, N, H sets are recalculated for the tag t and all vertices that contain t in their PL sets. If the attribute is added to a g-node, the schema designer can add the attribute to the $N_e$ sets of the vertices, for which the sets are recalculated, during the process of set recalculation.

2. **Removing an attribute from a tag t**. The deleted attribute is excluded from $N_e(t)$. The I, N, H sets are recalculated for the tag t and all vertices that contain t in their PL sets. Note that if a is the attribute being removed and $t \in P(s)$, $a \in N_e(s)$, then according to the axioms a is added to N(s).

3. **Adding a containment relation of a tag $t_2$ into a tag $t_1$ at a node s of $t_1$ s-tree**. The condition that $\text{G-root}(s) = t_1$ is checked to ensure that this operation is legal. If the condition fails the transformation is rejected by the system. s is added to $IN(t_2)$. If the acyclicity axiom is violated the changes should be rolled back and the transformation rejected.

4. **Removing a containment relation of a tag $t_2$ into a tag $t_1$ at a node s of $t_1$ s-tree**. The conditions that G-root(s) = $t_1$ and $s \in IN(t_2)$ are checked. If any of them fails the transformation is rejected by the system. s is removed from the system. s is removed from $IN_e(t_2)$ if it is present there.

5. **Adding an s-node $s_1$ to an s-tree as a child of $s_2$**. $IN(s_1)$ is set equal to $s_2$. $S = \{s' \mid s_2 \in IN(s')\}$. S is divided into two non-empty disjoint sets. If it is impossible the operation is rejected by the system. For elements of one of these disjoint sets, $s_1$ is added to their $IN_e$ sets. $s_1$ is added to *T*. If $s_2$ has '|' mark and the second disjoint set consists of one element the mark of $s_2$ is changed to ','.

6. **Removing an s-node $s_1$ from an s-tree as a child of $s_2$**. If $s_2$ is a g-node the operation is rejected. $S = \{s' \mid s_1 \in IN(s')\}$. If S contains a g-node the operation is rejected. Otherwise, s is removed from *T* and removed from the IN sets of elements of S. The marks for the new "edges" are calculated according to the edge-marking rule, that is, their marks are the products of the "edges", which they replace.

7. **Adding a g-node $t_1$**. The node is added to *T*. $IN_e(t_1)$ is associated with $t_1$. According to $IN_e(t)$ the IN(t) is calculated . An s-node s is added to *T*. $IN_e(s)$ is set to $t_1$. s is added to at least one $IN_e$ set as specified by schema designer, in simplest case it should be added to the end tag $IN_e$ set.

8. **Removing a g-node $t_1$**. If $t_1$ is a root the system rejects the removal. Otherwise, the removal of the g-node is done in three stages:

- All native attributes of the g-node are removed using 2.
- In a cycle while possible (that is, while the system accepts any transformation of the kind) perform the two actions: removing s-nodes of the s-tree of $t_1$ us ing 6 and removing containment relations using 3.
- The node and its s-tree are removed from *T*. Since containment relations were removed while possible there is one containment relation left. Let $t_2$ be the only node contained in $t_1$. $t_1$ is substituted with $t_2$ by making $IN_e(t_2)$ += $IN_e(t_1)$ before removing $t_1$ from *T*.

9. **Altering edge mark**. Schema designer explicitly alters mark on an edge.

**6. Related work.** The question of database schema evolution is raised for every database if it is used for a reasonable time span. For relational databases the problem is typically solved by DBMS manufacturers (such as Oracle), there are a number of systems for object-oriented ones (TIGUKAT [6], Orion [3]), but the problem for XML-databases is still open. As a rule, each XML-database designer supports its evolution manually or using "home-made" tools.

Object-oriented database looks rather similar to XML one because of their hierarchical structure. The evolution support for both databases uses the same basic principles. Orion system solves the schema evolution problem partially by using the set of restrictions (invariants) to the schema and a number of permitted operations over it. TIGUKAT offers a general solution and describes Orion invariants in terms of its axiomatic model.

The paper [1] introduces a basic axiomatic model of XML schema, which is an adaptation of the TIGUKAT model for semi-structured data. The model operates with tags and their inclusion without any attention to the rules of their inclusion. In that terms the model supports simple schema modifications and is presented as a basic tool for XML-schema evolution support. Here we extended this model by introducing regular expressions support mechanism, which allows working with schema completely matching original DTD.

**7. Conclusion.** The basic axiomatic model supports XML schema evolution in general case at the level of document data. The extended model proposed here works with the structure of schemata too. Using the axiomatic model one can control the schema modifications and maintain data consistency while the changes are made, thus, enabling a semi-automatic support for schema evolution.

The model suits well for the comparison of schemas. Basic schema changes provided in Section 5 may be used to describe a transformation of one schema into another. A transformation process may be considered as a schema evolution, so miscellaneous XML documents may be treated as evolved versions of one document if their schemas are transformed to the original one. The idea is helpful in various tasks like merging Web-databases or creating wrappers accessing semi-structured data.

**References:** **1**. *Coox S.* Axiomatization of Schema Evolution in XML-databases. // To appear in: Programmirovanie. 2003. No. 3. P. 1-9. (in Russian) **2.** *John E. Hopcroft, Jeffrey D. Ullman.* Introduction to Automata Theory, Languages and Computation. Addison-Wesley. 2000. ISBN 0-201-44124-1. **3.** *Won Kim, Hong-Tai Chou.* Versions of Schema for Object-Oriented Databases. // In proc. VLDB conf. 1988. P. 148-159. **4.** *N. Pitts.* XML in Record Time. Sybex. 1998. ISBN 0-7821-2340-6. **5.** *Randal J. Peters, M. Tamer Oszu.* Axiomatization of Dynamic Schema Evolution in ObjectBases. // In proc. ICDE. 1995. P. 156-164. **6.** *Randal J. Peters, M. Tamer Oszu.* An Axiomatic Model of Dynamic Schema Evolution in ObjectBase Systems. // ACM Transactions on Database Systems. 1997. № 22(1). P. 75-114.