Specification and Validation of the Referential Integrity Constraint in XML Databases

Jovana Vidaković*, Ivan Luković**, Slavica Kordić **
* University of Novi Sad, Faculty of Sciences, Department of Mathematics and Informatics, Novi Sad, Serbia
** University of Novi Sad, Faculty of Technical Sciences, Novi Sad, Serbia
jovana@uns.ac.rs, ivan@uns.ac.rs, slavica@uns.ac.rs

***Abstract*—The referential integrity constraint (RIC) is one of the key constraints that exists in every data model. Current XML database management systems (XML DBMSs) do not completely support this type of constraint. The structure of an XML document can contain the information about the relationship between elements, but modern XML DBMSs do not perform automatic RIC verification during database operations execution. It is therefore necessary to implement and perform additional actions to provide a fully functional validation of the referential integrity constraint. In this paper we discuss different structures of XML documents, and their impact on the referential integrity constraint. We also present two types of its implementation in XML databases: using XQuery functions and triggers. Those techniques correspond to the characteristics of a selected XML DBMS.**

I. Introduction

Referential integrity constraint (RIC) is one of the most important and most common constraints in every type of database management system. While it is completely supported with both specification and implementation in relational database management systems (RDBMSs), it is only partially covered in XML database management systems (XML DBMSs).

XML databases allow in a certain way specification and implementation of several most used constraints like keys, primary keys, foreign keys, and unique constraints. Those are the building blocks for the specification of the RIC. However, modern XML DBMSs do not support either specification, or an automatic validation of this constraint. It is possible to insert, update, or delete elements in DML DBMSs in a way to violate the RIC.

It is possible to validate a database against basic constraints, e.g. primary keys or foreign keys, on demand. However, it cannot automatically maintain the database consistence, as the data has already been changed at the moment of a constraint validation.

A goal of this paper is to propose an approach to the specification and implementation of the referential integrity constraint (RIC) under XML DBMSs. XML DBMSs show some limitations caused by the structure of XML documents describing a database schema. Since an XML document can be structured following the well-known rules in different ways [19], we discuss how a selected document structure can support XML constraints, and particularly RICs.

Apart from Introduction and Conclusion, the paper is organized as follows. Section 2 describes related work. In Section 3, approaches for structuring XML documents are discussed. The notion of RIC, as well as its validation, are presented in Section 4.

II. Related work

RIC is one of the constraints that exists in every data model. This constraint is very well supported in relational data model. There are numbers of papers dealing with this constraint [1], [2], [3]. It also exists in XML data model, as it is presented in [4] and [5]. In [6] and [7], the notions of XML inclusion dependency (XID) and XML foreign key (XFK) are introduced over Document Type Definition DTD.

Due to hierarchical structure of an XML document, a scope of the uniqueness of an element is to be defined. The uniqueness scope can be a whole document or only a part of the document. A specification of XML constraints starts with declaring XML relative and absolute keys [8], [9], [10].

An XML document specifying a database schema can have a hierarchical or relational structure [11]. A hierarchical structure is the one where relationships between elements are modelled by nested structures. If the relationship among entities is “many to many”, it yields unnecessary repeating of some elements and leads to unexpected redundancy. If the structure is relational, relationships between elements are expressed by using keys and foreign keys. Each element has its own identification that is used as a key value to be referenced from other elements via foreign keys. There are pros and contras of both the approaches. In a hierarchical XML document, the normalization cannot be applied. RIC cannot be applied either, as stated in [11]. Since this paper deals with RIC, we cannot consider the hierarchical XML document structure in our approach. By this, we adopt a relational structure of XML documents, where all elements are direct descendants of the root element. As a rule, such documents are smaller and with no redundancy [11]. Instead of nesting, elements are linked using foreign keys. In our approach, RICs are used over such structure.

In [12], the authors introduce an approach to the constraint taxonomy in the XML data model, by means of an analogy to a similar taxonomy in the relational data model. In this paper we also use the notion of RIC according to the taxonomy proposed in [12].

There are many papers dealing with RICs. Most of them propose approaches to their specification only, while the implementation is not considered.

We propose the implementations of RICs based on XQuery functions or triggers, depending on the trigger support level in a selected XML DBMSs. A typical way of constraint implementation in XML DBMSs is by means of triggers, as stated in [13], [14], and [15]. An XML DBMS supporting triggers in a way similar to relational databases is Sedna [16]. We have used Sedna for an implementation of the RIC by triggers. On the other hand, there are XML DBMSs with no appropriate support of triggers, such as eXist [17]. For those XML DBMSs, we propose a RIC implementation by XQuery functions [18].

III. Approaches to the Stuctures of an XML Document, Representing a Database Schema

One of the most important challenges in forming an XML document is defining relationships between elements. A relationship can be represented by physical positioning using nested structures. This is the hierarchical approach.

The relational approach of forming an XML document does not use nesting of elements. If two elements are related, keys and foreign keys are used to represent the relationship. Each element has identifier that can be used for referencing in another element using key and keyref elements.

Beside the data integrity, elements must be uniquely identified, the referential integrity constraint has to be satisfied, and there should not be update anomalies. If the hierarchical structure is used, a relationship between a parent element and a child element is represented only by physical positioning and the referential integrity constraint cannot be validated in that document.

If a relational structure is used, the key element is used to uniquely identify an element. The keyref element is used to connect a child element with its parent element. In this case, the referential integrity constraint is checked during validation, because each value in the keyref element must be an existing value in one key element.

The relational structure is better for use if the referential integrity constraint should be implemented using XML Schema document. Documents created using this kind of XML Schema document avoid redundancy, but they are larger. If the hierarchical structure is used, documents are more readable and clear, but foreign key cannot be declared and implemented in XML database. If an XML document is hierarchically structured and a child element is under its parent element, which represents "parent-child" relationship, then there is no foreign key in a child element to reference a parent element. Foreign key is necessary for referential integrity constraint.

The example of a hierarchical XML Schema document is given in Figure 1. An XML document created using this hierarchical schema is given in Figure 2. This schema describes the relationship between parents and children. A parent can have several children, but a child belongs to one parent. The element Child is dependent on the element Parent. The element Child is under the element Parent in the hierarchical structure. A position of the element Child represents its dependency on the element Parent. Each element Child has unique value of attribute child_id within its element Parent.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified"
attributeFormDefault="unqualified">
 <xs:element name="Root">
  <xs:complexType>
   <xs:sequence>
     <xs:element name="Parent" maxOccurs="unbounded">
      <xs:complexType>
       <xs:sequence>
        <xs:element name="Child" maxOccurs="unbounded">
         <xs:complexType>
          <xs:attribute name="child_id" type="xs:string"
use="required"/>
          <xs:attribute name="child_name"
type="xs:string" use="required"/>
         </xs:complexType>
        </xs:element>
       </xs:sequence>
       <xs:attribute name="parent_id" type="xs:string"
use="required"/>
       <xs:attribute name="parent_name" type="xs:string"
use="required"/>
       <xs:attribute name="parent_lastname"
type="xs:string" use="required"/>
      </xs:complexType>
     </xs:element>
    </xs:sequence>
  </xs:complexType>
 <xs:key name="Parent_PK">
  <xs:selector xpath="Parent"/>
  <xs:field xpath="@parent_id"/>
 </xs:key>
</xs:element>
</xs:schema>
```

Figure 1. Hierarchical XML Schema document

```
<?xml version="1.0" encoding="UTF-8"?>
<Root xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
 xsi:noNamespaceSchemaLocation="ParentChild.xsd">
 <Parent parent_lastname="Peric" parent_id="R1"
parent_name="Pera">
  <Child child_id="D1" name="Maja"/>
  <Child child_id="D2" name="Ana"/>
 </Parent>
 <Parent parent_lastname="Maric" parent_id ="R2"
parent_name="Mara">
  <Child child_id="D1" name="Marko"/>
 </Parent>
 <Parent parent_lastname="Ilic" parent_id ="R3"
parent_name="Ilija">
  <Child child_id="D1" name="Ivan"/>
 </Parent>
 <Parent parent_lastname="Antic" parent_id ="R4"
parent_name="Ana">
  <Child child_id="D1" name="Aca"/>
  <Child child_id="D2" name="Maja"/>
 </Parent>
</Root>
```

Figure 2. Hierarchical XML document

The key of the element Child is a relative key, since the values of the attribute child_id are unique only in its parent element. This key can be defined in the XML Schema document like it is presented in Figure 3.

```
<xs:key name="Child_PK">
        <xs:selector xpath="Child"/>
        <xs:field xpath="@child_id"/>
</xs:key>
```

Figure 3. Relative key of the element Child

If the element Child does not depend on the element Parent, i.e. it has its own key, that key is now absolute key. It is presented in Figure 4. The XPath expression Parent/Child denotes that value of the attribute @child_id in each element Child is unique within the whole document.

```
<xs:key name="Child_PK">
 <xs:selector xpath="Parent/Child"/>
 <xs:field xpath="@child_id"/>
</xs:key>
```

Figure 4. Absolute key of the element Child

XML documents can be structured in different ways: all data are placed in elements, all data are placed in attributes, or some of data are placed in elements and some in attributes. In this paper we used the second rule, i.e. all data are placed in attributes. We also adopted that an XML document has a root element and all other elements are on the same level under the root element.

This is important for the following example. If an element has more than one foreign key from different elements, physical positioning in hierarchical structure is not a good way for representing this kind of relationship. Attributes that represent foreign keys have to exist in the element. The relational structure is better for representing this document.

A relational XML Schema document is given in Figure 5. This schema decribes relationship between workers and projects in a company. One worker can work on many projects. On one project there can work many workers. For each worker assigned to a project, there is number of hours spent on that project (attribute hrs). Between elements Worker and Project, there is the relationship with cardinality “many to many”. Element Engagement represents the engagement of one worker on one project. It has foreign keys wId and pId. Every element Engagement is unique since its primary key is union of attributes wId and pId, which are foreign keys.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified"
attributeFormDefault="unqualified">
 <xs:element name="Work">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="Worker" maxOccurs="unbounded">
     <xs:complexType>
      <xs:attribute name="wId" type="xs:string"
use="required"/>
      <xs:attribute name="firstName" type="xs:string"
use="required"/>
      <xs:attribute name="lastName" type="xs:string"
use="required"/>
     </xs:complexType>
    </xs:element>
    <xs:element name="Project" maxOccurs="unbounded">
     <xs:complexType>
      <xs:attribute name="pId" type="xs:string"
use="required"/>
      <xs:attribute name="name" type="xs:string"
use="required"/>
     </xs:complexType>
    </xs:element>
    <xs:element name="Engagement"
maxOccurs="unbounded">
     <xs:complexType>
      <xs:attribute name="wId" type="xs:string"
use="required"/>
      <xs:attribute name="pId" type="xs:string"
use="required"/>
      <xs:attribute name="hrs" type="xs:int"
use="required"/>
     </xs:complexType>
    </xs:element>
   </xs:sequence>
  </xs:complexType>
  <xs:key name="Worker_PK">
   <xs:selector xpath="Worker"/>
   <xs:field xpath="@wId"/>
  </xs:key>
  <xs:key name="Project_PK">
   <xs:selector xpath="Project"/>
   <xs:field xpath="@pId"/>
  </xs:key>
  <xs:key name="Engagement_PK">
   <xs:selector xpath="Engagement"/>
   <xs:field xpath="@wId"/>
   <xs:field xpath="@pId"/>
  </xs:key>
  <xs:keyref name="Engagement_FK1" refer="Worker_PK">
   <xs:selector xpath="Engagement"/>
   <xs:field xpath="@wId"/>
  </xs:keyref>
  <xs:keyref name="Engagement_FK2" refer="Project_PK">
   <xs:selector xpath="Engagement"/>
   <xs:field xpath="@pId"/>
  </xs:keyref>
 </xs:element>
</xs:schema>
```

Figure 5. Relational XML Schema document

The XML document valid to the schema given in Figure 6. There is no redundancy and every key in this document is an absolute key. If a hierarchical structure was used in this example, it would lead to repeating of either element Worker or element Project.

```
<?xml version="1.0" encoding="UTF-8"?>
<Work xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:noNamespaceSchemaLocation="Work.xsd">
 <Worker wId="R1" firstName="Petar" lastName="Petric"/>
 <Worker wId="R2" firstName="Marko" lastName="Maric"/>
 <Worker wId="R3" firstName="Ana" lastName="Antic"/>
 <Worker wId="R4" firstName="Nina" lastName="Ninic"/>
 <Project pId="P1" name="Plate"/>
 <Project pId="P2" name="PDV"/>
 <Engagement wId="R1" pId="P1" hrs="10"/>
 <Engagement wId="R2" pId="P1" hrs="20"/>
</Work>
```

Figure 6. Relational XML document

IV. THE REFERENTIAL INTEGRITY CONSTRAINT IN XML DATABASES

In the XML data model, a referential integrity constraint can be modeled with keyref element in XML Schema document. This constraint can be violated by update operations. XML database management systems do not implement the referential integrity constraint completely because validation is not done after the operation but only on demand. If the validation is not requested it will not be done.

The taxonomy of constraint in XML databases is given in [12]. According to that taxonomy, the referential integrity constraint in XML data model is defined in the following way:

RICon(2,
me,
E1@X⊆E2@Y,
e1@X⊆e2@Y, e1∈E1, e2∈E2, Key(E2, Y),
((e_1, referencing, {(insert, noAction), (update, noAction)}),
(e_2, referenced, {(delete, noAction, cascade), (update, noAction, Cascade)})))

The specification scope of the referential integrity constraint is bounded by two element types. It is interpreted over 2 elements, so the interpretation scope is ‘more elements’ (me). The parametrized formula pattern for this constraint is E1@X⊆E2@Y, where E1 is a referencing element type, E2 is a referenced element type, and the set of attributes Y is the key of the element type E2. A validation rule, used to validate RIC is e1@X ⊆ e2@Y, where the element e1 is of the type E1, and the element e2 is of the type E2. Critical operations, which RIC can cause, are insert or update of the referencing element, and delete and update of the referenced element.

The validation of RIC is done on insert, update or delete operations, over either referencing or referenced element. Delete or update of the referenced element is not allowed (noAction) if that element is connected with at least one referencing element. Delete or update of the referenced element can be cascaded (cascade), which means that both referenced and referencing elements will be deleted or updated. Insert or update of referencing element is not allowed (noAction), if there is no corresponding referenced element.

This constraint cannot be implemented in the same way like in the relational databases. An XML document is validated against an XML Schema document and then the referential integrity constraint is checked. However, there is no mechanism to validate the constraint during update operations. XML database management systems, which we used, do not have support for cascade action or restrict action. Those actions should be implemented using procedural mechanisms. We used XQuery functions and triggers. Those techniques correspond to the characteristics of a selected XML DBMS. Since XML DBMS Sedna does not support XQuery functions, the referential integrity constraint is validated via triggers.

Let us validate the RIC over the XML document that is given in Figure 6. The RIC can be violated when updating or deleting a worker. The constraint validation can be implemented either by XQuery functions, or by triggers. XQuery functions are used in the eXist DBMS. The validation of an RIC constraint using XQuery functions is presented in Section IV.A. Triggers are used in Sedna DBMS. A validation based on triggers is presented in Section IV.B. We propose here the implementation methods for the RICs, using the mechanisms provided by the two representative XML DBMSs. We select eXist DBMS for the RIC implementation using XQuery functions, and Sedna DBMS for the implementation via triggers.

A. *RIC implementation for eXist*

eXist is one of the native XML database management systems [17]. eXist enables storing of XML documents and querying using XQuery language [18]. We used eXist DBMS to create functions that control the referential integrity constraint.

In this paper we describe deleting and updating of the referenced element, i.e. deleting and updating of the Worker element. The Worker element is part of the XML document, which is given in Figure 6. It can be implemented in two ways: by preventing (noAction) or by cascade action (Cascade). These actions have to be done procedurally because, to the best of our knowledge, XML databases do not support prevention or cascaded action.

XQuery function for deleting of the Worker element with preventing the operation, is given in Figure 7. Deleting of the Worker element can be forbidden if there is an Engagement element that referencing that Worker element. The Worker element can be deleted if there are no Engagement elements referencing to that Worker element. The function canDeleteWorker checks if the Worker element can be deleted. The function doDeleteWorker deletes the Worker element, if it is possible.

```
declare function local:canDeleteWorker($OLD as
element(Worker))
    as xs:boolean{
        let $r :=
not(exists(doc('Work.xml')/Work/Engagement[@wId=$OLD/@w
Id]) or not
(exists(doc('Work.xml')/Work/Worker[@wId=$OLD/@wId])))
        return ($r)
};

declare function local:doDeleteWorker($OLD as
element(Worker))
    as xs:boolean{
        let $i := update delete
doc('Work.xml')/Work/Worker[@wId=$OLD/@wId]
        return true()
};

declare function local:deleteWorker($OLD as
element(Worker))
    as xs:boolean{
        let $delWID := if(local:canDeleteWorker($OLD))
            then local:doDeleteWorker($OLD)
            else false()
    return $delWID
};
```

Figure 7. XQuery function for deleting a worker - NoAction

If the cascade deleting has to be done, all the Engagement elements that references to the Worker element, have to be deleted first. The function canDeleteWorker checks if there is the Worker element with the required value of the wId attribute. If it is found, the function doDeleteEngagement deletes all the Engagement elements that have the same value of the wId attribute. After that, the function doDeleteWorker deletes the Worker element. XQuery function for deleting the Worker element with cascaded action is given in the Figure 8.

```
declare function local:canDeleteWorker($OLD as
element(Worker))
    as xs:boolean{
        let $exists :=
exists(doc('Work.xml')/Work/Worker[@wId = $OLD/@wId])
        return ($exists)
};

declare function local:doDeleteWorker($OLD as
element(Worker))
    as xs:boolean{
        let $i := update delete
doc('Work.xml')/Work/Worker[@wId=$OLD/@wId]
        return true()
};

declare function local:doDeleteEngagement($OLD as
element(Worker))
    as xs:boolean{
        let $i := update delete
doc('Work.xml')/Work/Engagement[@wId=$OLD/@wId]
        return true()
};

declare function local:deleteWorker($OLD as
element(Worker))
    as xs:boolean{
        let $delWID := if(local:canDeleteWorker($OLD))
            then local:doDeleteEngagement($OLD) and
local:doDeleteWorker($OLD)
            else false()
    return $delWID
};
```

Figure 8. XQuery function for deleting a worker - Cascade

Updating of the Worker element is the operation that can violate RIC, too. If the value of the attribute wId, which is the key of the Worker element, does not exist in any of the child elements Engagement, update cannot be completed, as the referential integrity constraint will be violated. Beside that, the function canUpdateWorker checks if there is such Worker element with the same value of the wId attribute. If one of these conditions is satisfied, the function doUpdateWorker is called and the old Worker element is replaced with the new Worker element. Updating of the Worker element by preventing action is given in Figure 9.

```
declare function local:canUpdateWorker($OLD as
element(Worker), $NEW as element(Worker))
    as xs:boolean{
        let $exists :=
not(exists(doc('Work.xml')/Work/Engagement
                            [@wId=$OLD/@wId]) or
exists(doc('Work.xml')/Work/Worker[@wId=$NEW/@wId]))
   return ($exists)
};

declare function local:doUpdateWorker($OLD as
element(Worker), $NEW as element(Worker))
    as xs:boolean{
   let $i := update replace
doc('Work.xml')/Work/Worker[@wId=$OLD/@wId] with $NEW
        return true()
};

declare function local:updateWorker($OLD as
element(Worker), $NEW as element(Worker))
     as xs:boolean{
   let $updWID := if(local:canUpdateWorker($OLD, $NEW))
            then local:doUpdateWorker($OLD, $NEW)
            else false()
    return $updWID
};
```

Figure 9. XQuery function for updating a worker - NoAction

If update is cascaded, after updating the Worker element, every child Engagement element is updated too. The whole Engagement element with the matching wId value is replaced with the new element with the new wId value. The values of the attributes pId and hrs are the same as in the old Engagement element. Updating with cascaded action is given in Figure 10.

```
declare function local:canUpdateWorker($OLD as
element(Worker), $NEW as element(Worker))
    as xs:boolean{
        let $exists :=
exists(doc('Work.xml')/Work/Worker[@wId = $OLD/@wId])
and not (exists(doc('Work.xml')/Work/Worker[@wId =
$NEW/@wId]))
   return ($exists)
};

declare function local:doUpdateWorker($OLD as
element(Worker), $NEW as element(Worker))
    as xs:boolean{
        let $i := update replace
doc('Work.xml')//Worker[@wId=$OLD/@wId] with $NEW
        return true()
};

declare function local:doUpdateEngaement($OLD as
element(Worker), $NEW as element(Worker))
    as xs:boolean {
        let $r :=
        for $i in doc('Work.xml')/Work/Engagement[@wId
= $OLD/@wId]
         let $pId:=$i//@pId
         let $hrs:=$i//@hrs
         let $r := update replace
doc('Work.xml')/Work/Engagement[@wId=$OLD/@wId
and @pId=$pId  and @hrs=$hrs ] with
<Engagement wId="{$NEW/@wId}"  pId={$pId}  hrs={$hrs}/>
            return <res/>
        return true()
};

declare function local:updateWorker($OLD as
element(Worker), $NEW as element(Worker))
    as xs:boolean{
        let $updWID := if(local:canUpdateWorker($OLD,
$NEW))
            then local:doUpdateEngagement($OLD, $NEW)
and local:doUpdateWorker($OLD, $NEW)
            else false()
    return $updWID
};
```

Figure 10. XQuery function for updating a worker - Cascade

B. *RIC implementation for Sedna*

We select Sedna XML DBMS because of its full support of triggers. Sedna [16] is a native XML DBMS. It also offers a large number of services for managing XML documents: querying, modifying, ACID transactions, query optimization, etc. It supports W3C XQuery implementation. Sedna allows the use of either XQuery functions or triggers.

Triggers in Sedna do not provide cascade deleting. In this paper, triggers for deleting and updating prevention are given.

Deleting of the Worker element is possible only if it is not referenced in any Engagement element. Otherwise, or if such worker does not exists, deleting is not allowed. The trigger for Worker deleting is given in Figure 11.

```
CREATE TRIGGER "RefIntWorkerBeforeDelete"
BEFORE DELETE
ON collection('RefInt')/Work/Worker
FOR EACH NODE
DO {
        if(exists(fn:doc('Work', 'RefInt')/
             Work/Engagement[@wId=$OLD/@wId]) or
 not (exists(fn:doc('Work', 'RefInt')/
               Work/Worker[@wId=$OLD/@wId])))
        then
        error(xs:QName("RefIntWorkerBeforeDelete")," 
Can’t delete worker because wId is used in Engagement
")
        else
                        ($OLD);
}
```

Figure 11. Trigger that does not allow deleting

Update of the Worker element is not allowed if there is referencing Engagement element or there is no matching value of the wId attribute in any Worker element. In other cases, the old Worker element is replaced with the new Worker element. The trigger for Worker updating is given in Figure 12.

```
CREATE TRIGGER "RefIntWorkerBeforeUpdate"
BEFORE REPLACE
ON collection('RefInt')/Work/Worker
FOR EACH NODE
DO {
        if (exists(fn:doc('Work', 'RefInt')/
              Work/Engagement[@wId=$OLD/@wId]) or
     exists(fn:doc('Work', 'RefInt')/
              Work/Worker[@wId=$NEW/@wId]))
        then
        error(xs:QName("RefIntWorkerBeforeUpdate")," 
Can’t update worker because wId is used in Engagement
")
        else
                        ($NEW);
}
```

Figure 12. Trigger that does not allow update

V. CONCLUSION AND FUTURE WORK

In this paper we discuss different approaches to the structure of XML documents representing database schemas and their impact on the Referential Integrity Constraint (RIC). We find the relational XML structure as more suitable for specifying RICs.

Further on, we present two approaches to the RIC specification and implementation in XML databases. One is based on the usage of XQuery functions, while the other on triggers.

We find the approach based on triggers as better for the RIC implementation, if a selected XML DBMS offers the appropriate trigger support, as it is a case with the Sedna

DBMS. However, a support of triggers in the eXist DBMS is a quite basic, so it cannot be used for the RIC implementation. In this case, we have proposed a usage of the XQuery functions.

Future work will cover approaches to the specification and implementation of other types of constraints over various XML DBMSs. The process of generating XQuery functions and triggers for constraint validation could be fully automated and based on the paradigm of a model driven software engineering and the appropriate transformation specifications. A research work on a code generator development is in progress. This code generator would make database designer's and developer's job easier and free them from manual coding of XQuery functions and triggers for validating constraints.

ACKNOWLEDGMENT

The research presented in this paper was supported by Ministry of Education, Science and Technological Development of Republic of Serbia, Grant OI-174023.

REFERENCES

[1] Luković I, Ristić S, Mogin P, "On The Formal Specification of Database Schema Constraints", I Serbian – Hungarian Joint Symposium on Intelligent Systems, September 19-20, 2003, Subotica, Serbia and Montenegro, Proceedings, Polytechnical Engineering College, Subotica, Serbia and Montenegro, and Budapest Polytechnic, Budapest, Hungary, pp. 125-136

[2] Mogin P., Lukovic I., Govedarica M., Database Design Principles, 2nd Edition, University of Novi Sad, Faculty of Technical Sciences, Novi Sad, Serbia and Montenegro, 2004

[3] Lukovic I., Ristic S., Mogin P., On The Formal Speci¯cation of Database Schema Constraints. 1st Serbian-Hungarian Joint Symposium on Intelligent System SISY 2003, September 19-20, 2003, Subotica, Serbia and Montenegro, Proceedings, pp. 125-136.

[4] Fan, W., XML Constraints: Specification, Analysis, and Applications, DEXA, 2005, pp.805-809.

[5] Fan, W., Simeon, J., Integrity constraints for XML, PODS, 2000, pp.23-34.

[6] Shahriar, M.S., Liu, J., On Defining Referential Integrity for XML, International Symposium on Computer Science and its Applications, 13-15 Oct 2008. ISBN 978-0-7695-3428-2, DOI 10.1109/CSA.2008.36, pp. 86-91.

[7] Shahriar, M.S., Liu, J., Towards a Definition of Referential Integrity Constraints for XML, International Journal of Software Engineering and Its Applications, Vol. 3, No. 1, January 2009. pp. 69-82.

[8] Buneman, P., Fan, W., Simeon, J., Weinstein, S., Constraints for Semistructured Data and XML, SIGMOD Record, 2001, pp. 47-54.

[9] Buneman, P., Davidson, S., Fan, W., Hara, C., Tan, W., Keys for XML. Computer Networks, 2002, 39(5), pp. 473-487.

[10] Buneman, P., Davidson, S., Fan, W., Hara, C., Tan, W., Reasoning about Keys for XML, Information Systems, 28(8):1037-1063, 2003.

[11] Chaudhuri, N., Glace, J., Wilson, G., Hierarchical vs. Relational XML Schema Designs, A Study for the environmental Council of States, Report ECO41T1, http://www.exchangenetwork.net/dev_schema/schemadesigntype.pdf, June 2006

[12] Vidaković, J., Luković, I., Kordić, S., Specification and Implementation of the Inverse Referential Integrity Constraint in XML Databases, 7th Balkan Conference in Informatics, ACM New York, USA, ISBN 978-1-4503-3335-1, DOI: 10.1145/2801081.2801111

[13] Tatarinov, I., Ives, Z., Halevy, A., Weld, D., Updating XML, Proccedings of the ACM SIGMOD International Conference on Managenent of Data 2001, pp. 413-424

[14] Bonifati, A., Braga, D., Campi, A., Ceri, S., Active XQuery, ACM SIGMOD 2001 May 21-24, Santa Barbara, California, USA

[15] Grinev, M., Rekouts, M, Introducing Trigger Support for XML Database Systems, Proceedings of the Spring Young Researcher's Colloquium on Database and Inforation Systems SYRCoDIS, St. Petersburg, Russia, 2005

[16] Sedna, Native XML Database System, www.sedna.org

[17] eXist, http://exist-db.org/exist/apps/homepage/index.html

[18] XQuery, http://www.w3.org/TR/xquery/

[19] Vidaković, J., Specification and Validation of Constraints in XML Data Model, Ph.D. Thesis, Faculty of Technical Sciences, University of Novi Sad, 2015. (in Serbian)