V.K. Sahu et al., Int. J. Of Integrated Engineering Vol. 3 No. 1 (2011) p. 1-4

Enabling Business Rule oriented Constraint in XML Update Validation

Norfaradilla Wahid

Department of Computer Science and Engineering
La Trobe University, Australia

Received 21 June 2012; accepted , available online

Abstract: In the database update validation field, we are often unaware that there is a softer side of business rules, where it is hard to be expressed in the form of data constraint, as they require human decisions, etc. Even though it is not as critical as data constraint; but it may influence the accurateness of system data in their own way. The motivation of our research is to handle this type of rule, particularly for XML update validation. We have come out with the concept of dynamic constraint validation special for XML data, which is based from soft type business rules. We also have proposed a few different types of dynamic constraints and an algorithm to extract only necessary information for dynamic type of rules w.r.t. the validation operation. In this paper, we mainly focus only on the single node updating for single step transition, which will affect the dynamic constraint. The dynamic constraint is expressed using Schematron and should be flexible enough to sit on top of any existing XML database. The analysis shows that the algorithm is in polynomial efficiency.

Keywords: business rule, static constraint, dynamic constraint, XML update validation, Schematron.

1. Introduction

Business rules represent the primary means by which an organisation can direct its business, defining the operative way to reach its objectives and perform its actions; the rules may also include policies, standards and facts. Two common ways to enforce business rules are through application logic and data constraint e.g. embedded integrity constraints. The constraints act as a system-rule and always have to do with the integrity of data, which ensure that data of the system are constantly in valid condition.

1.1 Motivation

We are often unaware that there is a softer side of business rules. These types of rules are hard to be expressed in the form of data constraint, as they require human decisions, senses, conducts, etc. In data-recording environment, these rules are not as critical as the integrity constraint and are unsuitable to be handled as one; however, they can somehow influence the accurateness of data of the system in their own way. The motivation of our research is to handle the softer type of business rules, *w.r.t.* XML updates validation.

In XML research, a lot of efforts have been expended in expressing and maintaining integrity constraints. According to [12] there are two major types of constraints, i.e. structural constraint and semantic constraint. We believe that these two constraints are rather static than dynamic. A dynamic constraint is meant to express the condition that involves facts/requirements between two and more states during their transition within a given state space. As an example, *"a salary must never decrease"* can be read as the new salary (state) must be always higher than the previous salary (state). In a managerial context, dynamic constraint can be seen as representations of "real world" constraints and business rules [2]. We would like to suggest that, the dynamic constraint is a softer type of business rule; hence, it is unsuitable to be expressed in data layer of information system architecture. Therefore, dynamic constraint can be added on top of the database system.

Generally, dynamic constraint is far more important in Temporal Database research (for example [9], [11] and [18]) than non-temporal database. This is because temporal database needs a set of constraints to specify the requirements between sequences of state changes within the timeline. However, we have to agree that dynamic constraints have their own importance for non-temporal type of database, especially during updates validation. We suggest that this is an important issue to raise and to investigate further. The XML document in Figure 1 can be considered as an example to show the dynamic constraint.

Based on the example, "state" can exist randomly from the element and attribute collection. For example, there is a query `delete node $target//AccountDetail/ Type`, or a query `insert node data as first into $target//Account[1]`. In this example, the states that require changes are `Type` and `Account[1]`. However, it is fairly obvious that changing target `Account[1]` and `Type` in this case does not require us to check or to compare the current condition of target

*Corresponding author: nwahid@students.latrobe.edu.au

```
<?xml version="1.0" standalone="yes"?>
<BankAccounts>
    <Account><AccountDetail no = '1231'>
                    <Type>Saving</Type>
                    <OpenDate>11/04/1974</OpenDate>
          <Balance>55555.55</Balance></AccountDetail>
          <AccountDetail no = '1331'>
                    <Type>Credit</Type>
                    <OpenDate>11/04/1999</OpenDate>
                    <Balance>130.00</Balance>
                    <Limit>5000</Limit></AccountDetail>
          <AccountHolder ssn = '1212'>
                    <LastName>Doe</LastName>
                    <FirstName>John</FirstName>
                    <BasicSalary>1500</BasicSalary>
          <MaritalStatus>married</MaritalStatus>
          </AccountHolder></Account>
</BankAccounts>
```

Fig. 1 Account.xml

state with the new condition that is going to replace them, i.e. checking of a special property needs to be maintained. The query is intended to replace current value of `MaritalStatus= married` with a new value

Now, consider a query, `replace value of node $target//AccountHolder/[$ssn=1212]/MaritalStatus with single`. `MaritalStatus=single`. If we refer to system rules, there is nothing wrong with the changes.However, by human judgment it is impossible for someone who is married to go back to single status; logically the marital status can only be changed to divorced or widowed. Therefore, the update is inappropriate. From this example, `MaritalStatus` is the state, and we need to check whether the update violates the properties in order to make changes from the current to new state. This is the 'fact' to be maintained between the two states. It is not compulsory for one to maintain this kind of properties, but without knowing the consequence, this inaccuracy might lead to the problem of deducing false facts in the future.

It is shown that, dynamic constraint checking is necessary during XML update, even in non-temporal environment. To ensure dynamic constraint violation can be detected before an update is performed, we need to specify clearly that constraint is part of the validation process during updates.

Specifying dynamic constraints explicitly in XML database specification gives the advantage of supporting data relevance. In XML, XML Schema, DTD and Relax NG, there are a few common schema languages, which can be used to enforce integrity constraints [12]. However, these languages suffer from the drawback of expressing any constraint that is not meant for structural information. In order to support dynamic constraints, a language that can accommodate business rules is required.

1.2 Motivation

In this paper, we start by introducing the concept of dynamic constraint for XML data. The constraint that will be discussed is transition constraint, particularly in single transition as explained in [6] and [8]. While some of the previous works deal with dynamic constraints as non-database constraints in information systems (for example [1], [16], [22] and [10]), our work focuses on ways to represent the constraints in XML and the proposal of them to be part of the validation process during XML database updates. To prove the workability of the concept, we employ the capability of Schematron[13] to express the dynamic constraints; Schematron can function on top of any XML schema. As the main focus is on single transition constraints, we assume that we do not require any specific physical medium to store sequence information of states as is used in the temporal system. Nevertheless, we propose a special XML file for the dynamic constraint validation.

The rest of the paper is structured as the following. Section 2 discusses background of the study, the definition of single transition constraint and some related works. Section 3 discusses a few possible cases of single transition checking for single node updating *w.r.t.* `replace node` or `replace value of node` operators and `delete-insert` pairing; we present the proposed algorithm for dynamic constraint checking. Finally, in the last section, we present the implementation and analysis of the algorithm.

2. Dynamic Constraint in Literature

2.1 Background and related work

In the present study, a comprehensive combustion In the field of information systems and databases, the term integrity normally refers to the correctness or validity of stored data, as defined explicitly by integrity rules or integrity constraints [19]. Integrity is a very important property of information systems. Lack of integrity usually has negative consequences, which in some cases may lead to serious problems [20]. As has been quoted by [1], the terms integrity and integrity constraints are sometimes treated differently.

In [3], Motro defines the concept of integrity as two main components: valid and complete. We can say that a database has integrity if all its data are correct (valid), and if it contains all relevant data (it is complete). Based on the definition of Fratenali and Panaboschi in [4], integrity constraints can be divided into two classes: inherent and explicit. The former express restrictions due to the semantics of the data model, i.e. they are implicit in the database schema. The latter are arbitrary properties to be satisfied by the database that cannot be captured by schema restrictions. Constraints can also be distinguished as static and dynamic constraints [5].

From our observations in XML database research, the works only hover around the left branches of the integrity/integrity constraint definitions discussed above. This happens especially during database update validation. The validation process is usually accomplished by checking the inherent and static rules (of hard constraint) where it is meant for checking the *correctness* (validity) of database. Nevertheless, ensuring the *relevance* or *completeness* of data is also a goal to be achieved. The lack of it may cause the system to deduce facts that are not valid. As such, we need to ensure that

both couplings are achieved. Based on our study, we notice that explicit constraints and dynamic constraints come from softer type of business rules as explained in the introduction. Based on this motivation, we choose to propose a way to handle this type of business rules and propose to incorporate the rules to be part of XML validation process.

In [1], the author defines dynamic constraints as a way to express conditions that involve facts of two or more states of the database. On the other hand, a static constraint expresses state-independent properties that must hold in any state of the database. It depends only on the current state, and is independent of any previous state of the database [5]. Below are examples of static constraints and dynamic constraints:

Example 1.
(static) an employee's salary must be less than that of his manager
(dynamic) an employee's salary must never decrease

Example 2.
(static) a student must register at least a subject
(dynamic) a student that has dropped a subject and has not been immediately (in subsequence) reinstated, would not be allowed to be readmitted

Example 3.
(static) a person is allowed to get married
(dynamic) a person that is single, is only allowed to change his/her status to married

Example 4.
(static) every employee will get a chance to be promoted (change job rank)
(dynamic) an employee can possibly be promoted by using the following sequence:
trainee → junior → senior → assistant manager →manager

The most popular types of dynamic constraints that have been studied are general constraints and transition constraints. In the literature, there is no clear differentiation between 'general' and 'transition' constraints. Halpin in [7] states that dynamic constraint is actually the 'transition constraint' that restricts how the business may change to the new state. On the other hand, Fraternali and Paraboschi [4] state that transition constraint is a special type of dynamic integrity constraint.

In this paper, we would like to treat the former and the latter as one type of constraint; we call it single transition constraint and will deal with the problem by adding a business rule language on top of XML Schema. We follow the definition of single transition constraint as in [6].

Definition 1 Single transition constraint for XML is a constraint to restrict the change between an old state (the input of the transaction) and a new state (resulting from that transaction) of an XML node *w.r.t* content of the node. It specifies the condition stating the properties of the new state, including relation between the new state and the old state.

Let tree $T=(N, E, r, \Sigma, \lambda)$ be the XML tree in document; D. N is a set of node. $E \subseteq N \times N$ is a set of edges. r is the root node. Σ is the set of element names appearing in D. There is a subset of Σ which associate with attributes att . λ is labelling function which associates an element name with each node other than the root, where $\lambda: N - \{r\} \rightarrow \Sigma$.

There is a set of target state $\Phi = \Sigma_\Phi \cup att_\Phi$ where Σ_Φ and att_Φ are subset of Σ and att. Let X_i be a node in T. There is
a transition queried by XQuery, xq.

$$X_i \xrightarrow[c_a]{} X_i'$$

Where c_a is a constraint for the transition, and X_i has a content x_a. Note that c_a is constructed from an element of Φ. If $\Phi \neq \varnothing$, $x_a \neq null$ and $c_a \neq null$, and for any XPath, $p = r//X_i$, where $i \geq 1$, and p leads to x_a, then X_i' can only be true if $c(X_i)$ yields true.

As a matter of fact, we would like to include the problem of life-cycle constraint as in [2] to be in this category as well. This is because life-cycle restriction (as in Example 4) is also evaluated one state at a time.

Single transition constraint is very minimal and does not require any specific medium to hold the historical information, for instance by using historic schema in [21], auxiliary relation in [17], etc. We only require information held by existing state in order to accept the new state.

2.2 XQuery Update

XML Updates refer to the acts of modifying XML data through the operators by an XML manipulation language [12]. To support XML Updates, The XQuery Update Facility [15] has been designed to extend XQuery in order to facilitate updates to XML nodes. For instance, the XQuery Update Facility enables the following functions: insert new elements, delete elements, rename elements and replace the content of an element. The extension is due to the nature of XQuery 1.0 that is free from side effects, i.e. an XQuery expression cannot alter an XML node. In XQuery Updates, all modifications are performed as soon as the expression is entirely evaluated, i.e. there are no side effects until the end; the side effects of the former instructions do not appear with the execution of the current instructions. The following contains two examples of updates, deletion and insertion of `AccountDetail` from `Account.xml`:

```
let $source := doc('Account.xml')
return
(  delete node $source//Account[last()]/
     AccountDetail[1],
  insert node <AccountDetail no=1234>
     </AccountDetail> into $source//
     Account[last()]/
```

)

On the other hand, XML Update validation can be explained as a process of checking the correctness of each XML update towards each constraint of the database. It is performed to ensure that database states are always free from any violation, regardless of any updates. In the last decade, we have witnessed various research works in update validation of XML documents. Focus of the studies is on areas such as revalidation of XML documents within time-based, validation based on updating attempt, changes detection in XML documents, etc. These works cover mainly the structural part of the validation, but interests have also been noted in semantic validation (Refer to [12] for literature). Most of these constraints can be expressed by using XML schemas. However, these schema languages are not capable of expressing constraints related to business rules.

The Schematron assertion language provides a mechanism for making assertions about the validity of an XML document by using XPath expressions. There are six commonly used elements in a Schematron document: `schema`, `ns`, `pattern`, `rule`, `assert`, and `report`. ISO Schematron is a validation and reporting language, which is based on the presence or absence of XPath in one or more XML documents. It has an emphasis on human-understandability and is simple to use and implement. Schematron is an ISO standard frequently used to complement ISO RELAX NG grammars. In short, Schematron is capable of expressing more business-rule constraints and it is usually used on top of XML schema. Figure 2 shows an example of business rule expressed in Schematron language:

```
<?xml version="1.0" encoding="UTF-8"?>
<sch:schema
xmlns:sch="http://www.ascc.net/xml/schematron">
    <sch:pattern name="Bank Account Rules">
        <sch:rule context="AccountHolder">
           <sch:assert test="BasicSalary>500
">Basic salary must  bigger than
500</sch:assert>
        </sch:rule>
    </sch:pattern>
</sch:schema>
```

Fig. 2. Example of Schematron language

3. Propose Method for Dynamic Constraint Checking

3.1 Single node XML Update affecting dynamic constraint

The domain has been divided in to several As discussed in the previous section, our goal is to propose a method for dynamic constraint validation during XML Updates. For any dynamic constraints, particularly the transition constraint, XML nodes involved are either single or multiple nodes. In the former situation, it happens if the substitution behaviour queried by XQuery is on a single independent node. In the latter situation, we need to consider the association of the updating node with another node, for instance, dynamic dependency, aggregation, etc. In this paper, we restrict the work on single transition for single node updating.

Since dynamic constraint is not a hard constraint, we propose that, the validation result is treated more as a trigger rather than outright error warning as in structural validation. Because dynamic constraint is a soft constraint, it is good to subject it to final human judgment or confirmation before an update can be effected.

As recommended in [15], XQuery Update extension offers a few update operations: `insert`, `delete`, `replace` and `rename`. Since dynamic constraint deals with properties of 'current' and 'new' states, we only consider the `replace` and `delete-insert` pairing for our validation. This is because these operators have the behaviour of *substituting* the properties that are currently held by a state.

Replace operations can be achieved by using `replace node` or `replace value of node` operators. `replace value of node` is used to update value or content of a node. In this instance, the identity of the target node is preserved. Only its value or contents (for an element or a document) is replaced. On the other hand, to replace a node, we use the `replace node` operator..

Case by case scenario for single node updating.

We describe here a few possible cases that can exist deliberately during XML updates. There are two main cases that could appear: the leaf node and non-leaf node. On the leaf node problem, the checking is further differentiated based on the targeted node, i.e. element and attribute.

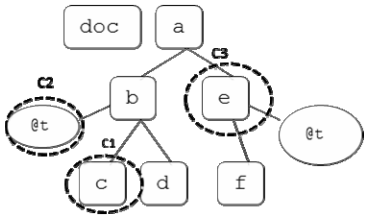

Fig.3. Example for case-based scenario

- *Case 1:* Refer to example C1 in Figure 3. Node `c` is a leaf element node. Say `c` contains a value `2`. Consider the following query update:

  `replace value of node a/b/c with "3"`

  In this case, the new state is "3" and the current state is "2". The idea is to make sure that replacing 2 with 3 will not cause violation against single transition constraint.

  `replace node a/b/c with <x> 3 </x>`

  Checking a `replace node` operator is also meant for checking the value contained in the replacing node, just like in `replace value of node` case; hence the operation is a bit tricky. This is because if the target is an element or a document node, all its former children will be removed and replaced. The replacing items are treated exactly as

the contents of a text constructor; all node items are replaced by their string-value. Even though the instruction is to replace a node and not a value, but it is possible for the replacing item to hold a value within it, which has a matching dynamic constraint; hence it could cause hidden violation.

- *Case 2*: Refer to example C2 in Figure 3 above. Node `t` is an attribute node and holds a value; hence it can be treated as a leaf node. Consider the following query update. Say `@t` content has a value "10"

  ```
  replace value of node a/b[@t]/@t with "12"
  ```

  and for `replace node` example,

  ```
  replace node $path/b[@t]/@t with @u=12
  ```

  Note that, for attribute type of node replacing, we only allow the replacing item to be of attribute type too. For this reason, validation with schema is required before dynamic constraint validation can be conducted. It will first decide whether the replacing item is from a valid name of namespace. The transition constraint checking will be treated exactly the same as for element node.
- *Case 3:* Refer to example C3 in Figure 3 above. Updating node `e` is also a bit tricky. Since `e` is a non-leaf, only `replace node` operation is allowed. Replacing a non-leaf node will replace all children nodes and will be handled as a string. However, it is also possible for the operation to implicitly involve value substitution.

  Say c content has a value of `2`

  ```
  replace node a/e with <g @t=1><f>New sub</f></g>
  ```

  This will yield a new sub-tree as requested.

  `<a><g @t=1><f>New sub</f<g>` in the above example.

  Notice that the query implicitly replaces a value held by f, which possibly is violating a transition constraint.
- *Case 4:* Replacement can also be seen as a combination of deletion and insertion. This method has been applied long before replace operator was introduced in XQuery as a new facility. The circumstances can happen in all cases 1 to 3 above. Let us consider the node in C1. For example, say we have the following query:

  ```
  delete nodes a/b/c,
  insert nodes <c>value</c> as last into a/b
  ```

These queries will be treated just like `replace node` or `replace value` as in previous cases.

3.2 Propose Algorithm for Single Transition Checking

The velocity, temperature and species mass fractions Figure 4 shows a generic view of how validation is handled. We begin the whole validation with a pre-processing module to generate a special XML file, which is meant to store only required data for validation. The process is completed by parsing the XQuery statements and extracting only necessary value, which in this case is any value *w.r.t* the operators in the previous section.

Generating the XML file needs the support of good XQuery Parser (see for example [14]). As query is executed, XQuery parser can be employed to produce the XML file. We propose the file to be simple and minimal; this file should be generic enough to work with different types of business-rule languages. In the following diagram, we called the file as `dyn_data.xml`.

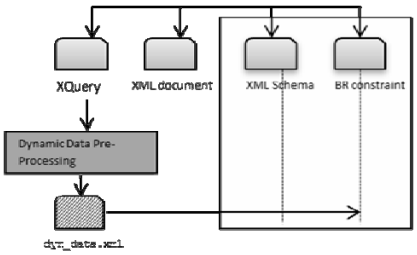

Fig. 4. XML Update validation flow

We proposed the file as in W3C XML Schema definition contained in Figure 5. `target_name` is the target element or attribute name, `new_state` is the new value requested by the query and `old_state` is the value extracted from original XML document.

```
<?xml version="1.0" encoding="utf-16"?>
<xsd:schema version="1.0"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="query">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="target_name">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="new_state"
minOccurs="0" />
              <xsd:element name="old_state"
minOccurs="0" />
            </xsd:sequence>
            <xsd:attribute name="path"
type="xsd:string" />
            <xsd:attribute name="file_name"
type="xsd:string" />
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Fig. 5. Lightweight XML for Single Transition Constraint

For our experiment, we use Schematron[13] as the business-rule language to express dynamic constraints. Figure 6 shows an example of Schematron document with transition constraints to be matched with XML document in Figure 1.

```
<?xml version="1.0" encoding="UTF-8"?>
<sch:schema
xmlns:sch="http://www.ascc.net/xml/schematron
">
    <sch:pattern name="Dynamic Constraint
Test">
          <sch:rule context="target">
               <sch:assert test="@name = 'age'
 and newstate > oldstate">Age can
 never
 decrease"</sch:assert></sch:rule>
      <sch:assert test= "name =
 'MaritalStatus' AND oldstate
```

```
        ='single' AND newstate='married'>
        Not possible </sch:assert>
            <sch:assert test= "name =
        'MaritalStatus' AND oldstate=
        'married' AND
        newstate=('divorce'|'widow')> Not
        possible </sch:assert>
            <sch:assert test= "name =
        'MaritalStatus' AND
        oldstate='divorce' AND
        newstate='married'> Not possible
        </sch:assert>
            <sch:assert test= "name =
        'BasicSalary' AND newstate>oldstate
        AND newstate>1000> Minimum salary
        not achieved </sch:assert>
    </sch:pattern>
</sch:schema>
```

Fig. 6. Example of Schematron rules

Meanwhile, the following is the proposed algorithm for single transition constraint validation. Let `Q`= update query, `D`= XML document and Scht= Schematron rule. While `t`=target name, i.e. element/attribute name, `p`= path expression of the target name, `f`= file_name, `ns`=new_state(new value to update), `os`= old_state(old value to replace), and `tn`= target name(for checking). For readability, we separate the steps to generate `dyn_data.xml` into sub-algorithm in Figure 8. But, for analysis purpose in the next subsection, we will treat the algorithm in Figure 7 and 8 as a one algorithm.

ALGORITHM: Dynamic Constraint Validation

```
1.  START
2.  Get Q, D[] and Scht.
3.  Let Q= q1, q2,…,qm where q1 to qm are the sub
    queries of Q
4.  Set i = 0
5.  Generate dyn_data.xml (refer to Figure 8)
6.  VALIDATE dyn_data.xml with Scht
7.  IF Result = true
8.    Proceed update
9.  ELSE
10.   Trigger warning
11.   IF user choose true
12.   Proceed update
13.   ELSE
14.   Cancel update
15.   END-IF
16. END-IF
17. END
```

Fig. 7. Proposed algorithm for single transition constraint checking

Extracting data for `replace` operations is quite straightforward. In the case of `delete-insert` pairing, we test if there is any `delete` operation in the query. If there is any, we then test if any `insert` operation exists in the same query. We store these two pieces of information into `Del[]` and `Ins[]` which can be compared (if only both are not `null`). If there is any equality between both data, then it will also be treated as dynamic data and related information will be extracted from the original XML document. This strategy is used, as the nature of XQuery update is to evaluate all the sub-queries, and then perform all the updates in batches. Therefore, the strategy will provide fair and general evaluation on all sub-queries because they might not appear in the straightforward sequences.

SUB-ALGORITHM: Generating `dyn_data.xml`

```
1.  WHILE i<= m DO // XQuery parser usage

2.    IF qi contains 'replace value of node'

3.        Extract t, f, p, ns
4.      Set s1= t, f, p, ns
5.      Write s1 into file x

6.    ELSE IF qi contains 'replace node'

7.      Extract p and t
8.      Get tn based on p    //Get the real
          containing node
9.        IF (tn == t)
10.           Extract t, f, p, ns
11.       Set s2= t, f, p, ns
12.       Write s2 into file x
13.       END-IF

14.   ELSE IF qi contains 'delete node'

15.     IF Q contains 'insert node'
16.      Add into Del[] ← t, p
17.     END-IF

18.   ELSE IF qi contains 'insert node'

19.     IF Q contains 'delete node'
20.         Extract t, f, p, ns
21.       Write s3_temp = t, f, p, ns into file y
22.       Add into Ins[] ← t,p
23.     END-IF
24.   END-IF
25.   i++
26. END-WHILE //end XQuery parser usage
27. IF (Del[]!null && Ins[]!null)
28.   j= 0 // Delete- Insert pairing started
29.   WHILE j < SIZE OF Del[], DO
30.     Get p from ej of Del[]
31.     Add to DelList ← p
32.     j++
33.   END-WHILE
34.   k =0
35.   WHILE k < SIZE OF Ins[], DO
36.     Get p from ek of Ins[]
37.     Add to InsList ← p
38.     k++
39.   END-WHILE
40.   FOR i←0 to i< DelList size DO
41.     FOR j←0 to j< InsList size DO
42.       IF element ei equal to ej
43.         k ←0
44.         WHILE !EOF of y, DO
45.           IF k==j
46.             Read line k of file y = s3
47.             Write s3 into file x
48.           END-IF
49.         END-WHILE
50.       END-IF
51.       j++
52.     END-FOR
53.     i++
54.   END-FOR
55. END-IF //end pairing
56. WHILE (!EOF of x)
57.   Read t,f,p,ns
58.   FOR i← 0 to SIZE OF D[], DO
59.     Get os based on p
60.     Transform into related element and
        attribute,
        XML=TransformToXML(t, f, p, ns, os)
61.     i++
62.   END-FOR
63. END-WHILE
64. Write XML into dyn data.xml
```

Fig. 8. Proposed algorithm to generate `dyn_data.xml`

With regard to the cases discussed in the previous section, we can see that if a query forms a sequence of nodes to be updated, the algorithm will always extract the data *w.r.t* the internal content of nodes.

Assuming that there is an XQuery statement to replace values of `MaritalStatus` and `BasicSalary` as in Section I, the `dyn_data.xml` is as follows.

```
<?xml version="1.0" encoding="UTF-8"
standalone="no"?>
<queries>
  <target file_name="Account.xml"
name="MaritalStatus"
path="$target//AccountHolder/[$ssn=1212]/Mari
talStatus">
    <oldvalue>married</oldvalue>
    <newvalue>single</newvalue>
  </target>
  <target file_name="Account.xml"
name="MaritalStatus"
path="$target//AccountHolder/[$ssn=1212]/Basi
cSalary">
    <oldvalue>1500</oldvalue>
    <newvalue>900</newvalue>
  </target>
</queries>
```

Fig. 9. Example of generated dyn_data.xml

The `dyn_data.xml` in Figure 9 will be checked against Schematron file as in Figure 6 in order to identify any violation of single transition constraint. In the proposed algorithm, we show the transformation of XQuery Update statements into a proper input is achieved by getting only necessary value for dynamic constraint checking. This is the main contribution of our proposed algorithm. This generic data can be used with any business-rule languages. Since Schematron can exist independent of any other schema language, we use this language on top of XML Schema.

3.3 Experimental Setup

For experiment purpose, we run the program on Java platform with the support of BaseX[14] for the XQuery parser. The experiment is conducted in 2.3GHz Intel Core i5 computer running Windows XP. We use five synthetic XML datasets of varying sizes and store the dynamic constraints in a single Schematron file. Then, we run various update queries for each dataset. Examples of the XML data and the related file are shown in Figure 1, 6 and 9.

3.4 Analysis

Two important aspects of the algorithm efficiency are the amount of time required to execute the algorithm and the memory space it consumes. To analyse the algorithm, we calculate the performance by applying run-time complexity of Big-O notation and by means of worst-case scenario. By using this calculation, we can see how the run-time will grow as the number of input N grows. The strategy is by dividing the algorithm into fundamental operations, i.e. the loops and other statements. By dividing the analysis fragment by fragment, we can evaluate the part that needs to be improved in the algorithm. Table 1 shows the summary of computational complexity calculation. By the sum and product rules, the total cost becomes,

Total cost, $f(N)$ = O(N)+O(N)+ O(N^3)+ O(N^2)+O(N), where it can be simplified into O(N^3) for worst case scenario. Fragments (a) to (d) basically come from sub-algorithm in Figure 8. The fragment that contributes to the most computational time is fragment (c). Figure 10 shows the worst-case performance of our algorithm. The dashing lines show the performance of each fundamental operation as in Table 1 while the thick line shows the overall performance of $f(N)$. Number of queries and number of documents both give impact to the number of N. We can say that as number of N increases, the required time for execution increases in cubic growth.

Table 1. Summary Of Asymptotic Complexity

Fundamental Operation	**Summary of complexity in worst case scenario.**
WHILE-LOOP *(a)*	The first while-loop on lines 1-26 costs 18N+2 which is in O(N) asymptotic time.
WHILE-LOOP *(b)*	The second while loop on lines 29-39 costs 8N+6 which is in O(N) asymptotic time.
FOR-LOOP *(c)*	The for-loop on lines 40-54 costs n^3+3n^2+4n+2 time i.e. O(N^3).
WHILE-LOOP *(d)*	The last while-loop on lines 56-63 costs $4N^2+4$ i.e. O(N^2).
OTHER STATEMENTS *(e)*	Other statements out of the loop costs 12+N, i.e. O(N) .

The constraints that we have discussed in the sections above are with the assumption that all the updates are based on all the naming convention as standardised in the `namespace`. However in real situations, referring to a `namespaces` is not compulsory. Someone could possibly have the intention to update or change node with a different name, as if it has the same intended meaning as the name before. For example, say we have a query `replace node $path/c with <x> 3</x>, it is` noticed that node `c` has been replaced with `x`. If the intended meaning of node `x` is actually the same as `c`, dynamic constraint still needs to be applied (if any). This problem is beyond the capability of what have discussed in this paper.

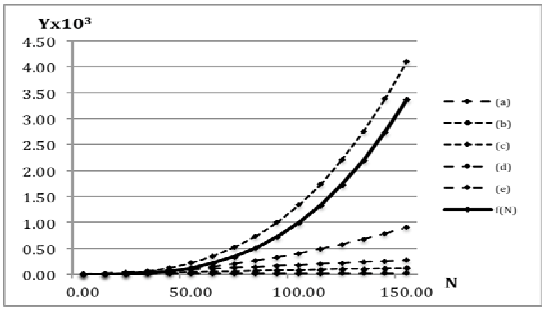

Figure 1. Asymptotic run-time performance

DISCUSSION

An issue that needs to be raised is that, the above algorithm actually extracts all data related to replace and delete-insert operation with the assumption that all might have a matching dynamic constraint. Hence, it could possibly deal with unnecessary data. Another issue is that current algorithm could not deal with indirect states transitions. For example, between the transitions of `s1` into `s2`, there are actually a few sub-queries that need to be performed in the middle before `s2` could be achieved. Our dynamic constraint data format is not minimal enough for this type of problem.

As a matter of fact, it is quite possible to operate the same validation via procedural trigger at the data level of a system. However, we believe that our proposed algorithm is much more significant in terms of organising the rules and it is less costly for larger size of data. In some situations, dynamic constraints can be derived from company policies and conflicts may occur between users who have the same roles. The policy with a soft rule characteristic is a flexible policy that can be modified depending on user's current situation. In contrast, hard rules cannot be modified. It is difficult to handle this problem through data level validation.

4. Future Work

Our most recent focus is in defining as many as possible dynamic constraints for XML Update validation. We are currently working on handling multiple nodes update and we called it as cumulative node constraint. With some extension of definition of this constraint, we manage to come out with another constraint i.e. dynamic Inclusion Dependency. And the main focus at the time being is to proposed an algorithm based on object identification technique to identify cumulative nodes constraint for the validation purpose.

5. Summary

One way to enforce business rules in XML applications is through XML data constraints, which can be static or dynamic. There are several works that deal with the maintenance of static constraints in the area of XML applications and XML databases. However, very few works have been done for the maintenance of dynamic constraints. In this paper, we propose a method to handle XML dynamic constraints during XML updates w.r.t single transition constraint on single node updating. To support the validation, we propose a lightweight XML file to store only required data for the validation, i.e. states that need to be changed are extracted from XQuery and original XML documents. We prove the concept of transition constraint validation by Java experiment with the support of BaseX. Based on the experiment, we find that the validation is particularly useful especially in ensuring the data conform to the business rules identified in the business rules schema data during updates. The analysis shows that our algorithm is in polynomial efficiency but it can be improved in the future for better polynomial function. For future works, we would like to extend the study to cover more types of dynamic constraints. We are currently working on the effects of single transition constraint in relation to multiple XML nodes *w.r.t* any node dependencies.

Acknowledgement

Contents of this paper will appear in the Proceeding of the 15th International Conference on Network-Based Information Systems, Melbourne in September 2012.

References

[1] M.A. Pacheco, Dynamic integrity constraints definition and enforcement in databases: a classification framework, Proc. of the First Working Conference on Integrity and Internal Control in Information System, Zurich, Switzerland, 1997, pp 65-87.

[2] B.O.D. Brock, A General Treatment of Dynamic Integrity Constraints., Data Knowl. Eng., 32(3), 2000, pp.223-46.

[3] A. Motro, Integrity = validity + completeness., ACM Transactions on Database Systems, 14(4), 1989, pp. 480-502.

[4] P. Fraternali and S. Paraboschi, A Review of Repairing Techniques for Integrity Maintenance, Proc. of the First International Workshop on Rules in Database Systems , Edinburgh, Scotlan, , 1993, pp.333-46.

[5] D. Plexousakis, Semantic integrity enforcement in knowledge bases, PhD. Qualifying Examination Paper, Department of Computer Science, University of Toronto, Canada, 1991.

[6] H. Balsters, and T. Halpin, Formal semantics of dynamic rules in ORM, Lecture Notes in Computer Science, 5333, pp. 699–708.

[7] T. Halpin, Fact-Orientation Meets Agent-Orientation, Proc. of Sixtth International Bi-Conference Workshop Agent-Oriented Information Systems II, Riga, Latvia, 2004, pp.97-109.

[8] H. Balsters, A. Carver, T. Halpin and T. Morgan, Modeling Dynamic Rules in ORM, Lecture Notes in Computer Science, 4278, pp. 1201-10.

[9] J. Chomicki and D. Toman, Implementing Temporal Integrity Constraints Using an Active DBMS, IEEE Trans. Knowl. Data Eng. 7(4), 1995, pp.566-82.

[10] M. Iwaihara, Supporting Dynamic Constraints for Commerce Negotiations, Proc. Second International Workshop on Advance Issues of E-Commerce and Web-Based Information Systems, Milpitas, California, pp.12-20, 2000.

[11] F. Rizzolo and A. A.Vaisman, Temporal XML: modeling, indexing, and query processing, VLDB J., 17(5), 2008, pp. 1179-12.

[12] N. Wahid, and E. Pardede, XML Semantic Constraint Validation for XML Updates: A Survey, Proc. of International Conference on Semantic Technology and Information Retrieval, Malaysia, 2011, pp.57-63.

[13] Schematron, http://www.schematron.com/

[14] BaseX, http://www.basex.org

[15] XQuery Update Facility, http://www.w3.org/TR/xquery-update-10/

[16] A.D. Sarma, A. Jain and C. Yu, Dynamic relationship and event discovery, Proc. of the Forth International Conference on Web Search and Web Data Mining, Hong Kong, 2011, pp. 207-16.

[17] J. Chomicki, History-less Checking of Dynamic Integrity Constraints, Proc. of the Eighth International Conference on Data Engineering, Tempe, Arizon, 1992, pp. 557-64.

[18] A. Artale, C. Parent and S. Spaccapietra, Evolving objects in temporal information systems, Annals of Mathematics and Artificial Intelligence 50(1-2), 2007, pp. 5–38.

[19] P. Grefen and P. Apers, Integrity control in relational database systems - an overview, Data Knowl. Eng., 10(2), 193, pp. 187-223.

[20] A. Olivé, Integrity constraints specification, Technical report LSI, Universitat Politècnica de Catalunya, Barcelona, Spain, 1995.

[21] D. Gubiani and A. Montanari, A relational encoding of a conceptual model with multiple temporal dimensions, Proc. of the Twentieth International Conference on Database and Expert Systems Applications, Linz, Austria, 2009, pp.792-806.

[22] M.A. Cibrán and B. Verheecke, Dynamic Business Rules for Web Service Composition, Proc. of the Second Dynamic Aspects Workshop, Chicago, USA, 2005, pp.13-8.