RULE-BASED QUERY TREE EVALUATION OVER FRAGMENTED XML DOCUMENTS

Ron G. McFadyen[+], Yangjun Chen[*]
Department of Applied Computer Science, University of Winnipeg,
515 Portage Avenue, Winnipeg, Manitoba, Canada, R3B 2E9

[+]*Supported by NSERC 105709-03 (139988) (Natural Sciences and Engineering Council of Canada)*
[*]*Supported by NSERC 239074-01 (242523) (Natural Sciences and Engineering Council of Canada)*

Keywords: XML, business rule, query tree, query evaluation, document fragmentation and assembly.

Abstract: XML documents are used to hold information and to make exchanges between systems. In this paper, we consider documents that embed knowledge and rules, which may contain considerable redundancy. To control redundancy and to provide for efficient execution of queries, documents are decomposed into fragments that are stored separately. Then, to materialize documents for end-users, they need to be dynamically constructed from their sources (separately stored fragments) by evaluating rules, which requires database queries to be executed according to the document structure.

1 INTRODUCTION

XML and related technologies are used extensively for storing, managing, and exchanging information. Our work is motivated by the consideration of documents describing requirements or business rules to be met to achieve some designation or status. According to (Business Rules Group 2000, Ross 1997), a business rule is a statement that defines or constrains some aspect of the business and is used to control the behavior of the business. For example, in a university calendar, we have rules that specify the valid collection of courses for a student to obtain a certain university degree.

As shown in (McFadyen et al 2005), such rules can be organized into a special structure, the so-called *synthesized query tree* (SQT), to govern their execution. An observation shows that such trees may heavily overlap. Then, the separate storage of SQTs not only leads to redundant space, but also causes repeated evaluation of rules. For this reason, we fragment individual SQTs and organize the sub-SQTs into a hyper tree structure, in which a leaf node can be a simple query or a subtree itself. This kind of organization is similar to document fragmentation (Salminen and Tompa 2001, Quint and Vatton 2004). However, in our case, a fragment is a subtree representing a set of rules, which produces a piece of a document dynamically. Our method also shares the flavour of active XML documents (Abiteboul et al 2002, Abiteboul et al 2003, Bonifati et al 2001), by which parts of the contents are generated by invoking a program or a web service; but differs from these in that we are concerned with the processing of business rules, which are evaluated along a tree structure in a bottom-up way.

The remainder of this paper is organized as follows. Next in section 2, we present the background information on requirement documents and synthesized query trees. In section 3, we present the SQT fragmentation based on the so-called *virtual SQTs*, which enable us to efficiently evaluate queries. Section 4 presents a short conclusion and directions for further work.

G. McFadyen R. and Chen Y. (2006).
RULE-BASED QUERY TREE EVALUATION OVER FRAGMENTED XML DOCUMENTS.
In *Proceedings of WEBIST 2006 - Second International Conference on Web Information Systems and Technologies - Internet Technology / Web Interface and Applications*, pages 199-205
DOI: 10.5220/0001237501990205

2 REQUIREMENTS DOCUMENT AND SQT STRUCTURE

In Figure 1, we show some typical majors found in a university calendar. Their XML representation is illustrated in Figure 2. In Figure 1, each document is a requirement for graduation from a certain major.

```
3-Year BSc (Geography)
  Graduation Requirement
      90 credit hours
  Residence Requirement
      Degree: minimum 30 credit hours
      Major: minimum 18 credit hours
  General Degree Requirement
      Humanities: 12 credit hours
      Science: 6 credit hours
  Major Requirement
      Minimum 30 credit hours
      Maximum 48 credit hours
      Required Courses
          23.202 Intro Geography I
          23.203 Intro Geography II
          23.331 Advanced Geography
      Choice
          23.205 Atmos Sci or 23.206 Earth Sci
```

a) Graduation Requirements - Geography

```
3-Year BSc (Physics)
  Graduation Requirement
      90 credit hours
  Residence Requirement
      Degree: minimum 30 credit hours
      Major: minimum 18 credit hours
  General Degree Requirement
      Humanities: 12 credit hours
      Science: 6 credit hours
  Major Requirement
      Minimum 36 credit hours
      Maximum 54 credit hours
      Required Courses
          44.101 Intro Physics
          44.203 Mechanics
          44.331 Relativity
```

b) Graduation Requirements - Physics

Figure 1: Graduation Requirements.

For instance, in Figure 1(a), we specify that to graduate with a 3-Year BSc in Geography a student must satisfy all of: *Graduation Requirement* (completion of 90 credit hours), *Residence Requirement* (completion of a minimum of 30 credit hours at the university and a minimum of 18 credit hours in Geography at the university), a *General Requirement* (12 credit hours in Humanities subjects and 6 credit hours in Science subjects), and a *Major Requirement* for certain courses in Geography. The fact that all four of these requirements must be met simultaneously is indicated by the attribute: *combining="AND"*, associated with the element GeographyRule in the second line of the document shown in Figure 2.

```
<GeographyRule title="Degree Requirement for 3-Year BSc(Geography)"
        combining="AND">
<GraduationRule title="Graduation Requirement"
        display="90 credit hours"  query= "... " ...>
</GraduationRule>
<ResidenceRule title = "Residence Requirement"
        combining = "AND" >
  <DegreeRule   title = "Degree"
          display = "minimum 30 credit hours"
          query = "SELECT sum(creditHours)
                FROM studentHistory
                WHERE studentNum = x
                  and institution="UW"
                  GROUP BY studentNum
                HAVING sum(creditHours) > 30" >
  </DegreeRule>
  <MajorRule  title="Major"  display="minimum 18 credit hours"
          query="... " ...>
  </MajorRule>
</ResidenceRule>
<GeneralRule      title="General Degree Requirement"
          combining="AND">
  <HumanitiesRule  title="Humanities"
          display="12 credit hours" query="... " ...>
  </HumanitiesRule>
  <ScienceRule  title="Science" display="6 credit hours" query="... "...>
  </ScienceRule>
</GeneralRule>
<MajorRule>      title="Major Requirement"
          combining="AND">
  <MinMaxRule
      display="Minimum 30 credit hours, Maximum 48 credit hours"
      query="... " ...>
  </MinMaxRule>
  <ReqCoursesRule   title="Required Courses"
          combining="AND">
    <Course
      display="23.202 Intro Geography I"  query="..." ...> </Course>
    <Course
      display="23.203 Intro Geography II"  query="..." ...> </Course>
    <Course
    display="23.331 Advanced Geography"  query="..." ...> </Course>
  </ReqCoursesRule>
  <ChoiceRule   title="Choice"
          display="23.205 Atmos Sci or 23.206 Earth Sci"
          combining="OR">
      <Course  query="..."  ...></Course>
      <Course  query="..."  ...></Course>
  </ChoiceRule>
</MajorRule>
</GeographyRule>
```

Figure 2: XML for 3-Year BSc (Geography).

As discussed in (McFadyen et al 2005), we can construct a synthesized query tree over such a document. (McFadyen et al 2005) presents two such tree structures: the boolean and the general SQT; for simplicity, here we only discuss the boolean SQT.

Definition 1: a *boolean synthesized query tree (BSQT)* is a tree where each leaf node v is associated with a boolean query $Q(v)$, and each internal node v is labelled with a tag *T(v)*, and an operator $\theta = \vee$ or $\wedge$; and each node v is assigned a boolean value, *V(v)*, determined as follows:

a) for a leaf node, $V(v)$ is *true* if the return value of $Q(v)$ is not empty; otherwise, it is *false*, and
b) for an internal node, with children v_1, ... v_n, $V(v) = V(v_1)\theta V(v_2)\theta\ldots\theta V(v_n)$.

For instance, for the graduation requirement of Geography, we will construct a tree structure as shown in Figure 3, which represents the rules, queries and relationships corresponding to the requirements shown in Figure 2. One of the defining characteristics of a query tree is that database queries are only present in leaf nodes.

To determine if a student can graduate, it is necessary to evaluate the appropriate SQT and its queries in the context of the student. Thus, to do this for a number of students, we have to traverse the SQT trees repeatedly, each time for a single student.

An observation of the queries present in the leaves shows that a slight modification will facilitate set processing. To see this, let's have a look at the query Q_1 shown in Figure 3. If we remove the condition studentNum = *x* from the where-clause, the execution of the query will find all the students with grade point ≥ 1 and more than 90 credit hours.

Now, for a given set of students, to check whether they are eligible to get a degree, we traverse the query tree bottom-up. During this process, all the queries attached with the leaf nodes are evaluated against the student records and the results are transferred to the internal nodes for further checking the specified logic conditions.

For students of a different major, a different SQT will be instantiated and traversed. Obviously, if two SQTs share a common subtree, this subtree will be traversed two times and its queries executed twice. For example, the General requirement in the document shown in Figure 1(a) is completely the same as the General requirement shown in Figure 1(b). If we evaluate the General subtree on its own it is traversed once and its queries are executed once. The returned result is then separated according to student majors and transferred to the parent in the respective SQTs. In this way, the common subtree is evaluated only once. This observation leads to the SQT fragments discussed in the next section.

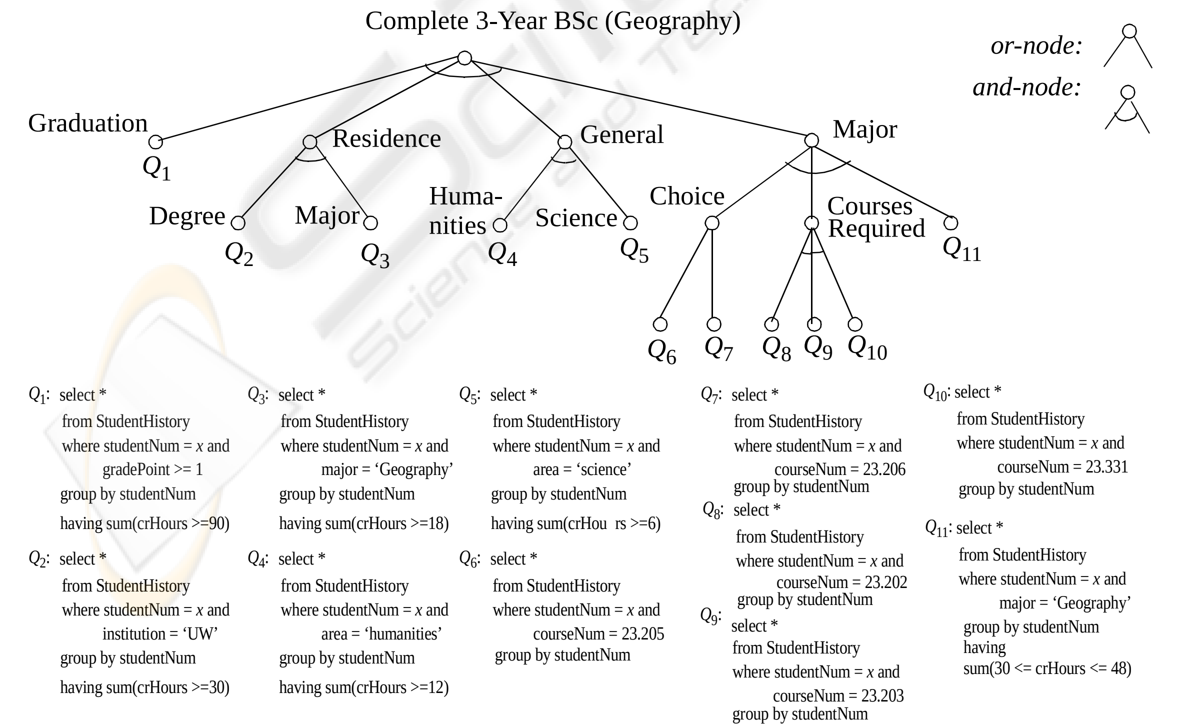


Figure 3: A Boolean SQT for graduation requirements.

3 DOCUMENTS: FRAGMENTATION AND EVALUATION

In this section, we discuss SQT fragmentation. First, we show how to fragment SQTs in 3.1. Then, we discuss how to evaluate fragmented SQTs in 3.2.

3.1 SQT Fragmentation

In (McFadyen et al 2005), two kinds of SQTs: boolean SQTs and general SQTs are defined. Both can be fragmented to speed up query evaluation. For simplicity, however, we show only how to fragment the boolean SQTs and the general SQTs can be handled in a similar way.

First, we introduce the concept of *virtual boolean synthesized query trees*, based on which the boolean SQT fragmentation is conducted.

Definition 2: a *virtual boolean synthesized query tree (VBSQT)* is a tree where a leaf node v is either

a) associated with a boolean query $Q(v)$, or
b) specifies a fragment that is another *VBSQT* (such a leaf node is called a virtual leaf node),

and each internal node v is labelled with a tag *T(v)*, and an operator $\theta = \vee$ or $\wedge$; and each node v is assigned a boolean value, *V(v)*, determined as follows:

a) for a leaf node that is a query, $V(v)$ is *true* if the return value of $Q(v)$ is not empty; otherwise, it is *false*, and
b) for a leaf node that specifies a fragment, $V(v)$ is the value of the fragment, and
c) for an internal node, with children v_1, ... v_n,
$V(v) = V(v_1)\theta V(v_2)\theta\ldots\theta V(v_n)$.

For instance, the tree shown in Figure 4(a) is a virtual version of the tree shown in Figure 3, in which the leaf nodes labelled with v_1, v_2, and v_3 represent the three trees shown in Figure 4(b), respectively. They are singled out since they also belong to other SQTs. To see this, examine the tree shown in Figure 4(c), which is a VBSQT for the 3-Year BSc in Physics and where v_1, v_2, and v_3 are three of its leaf nodes, too.

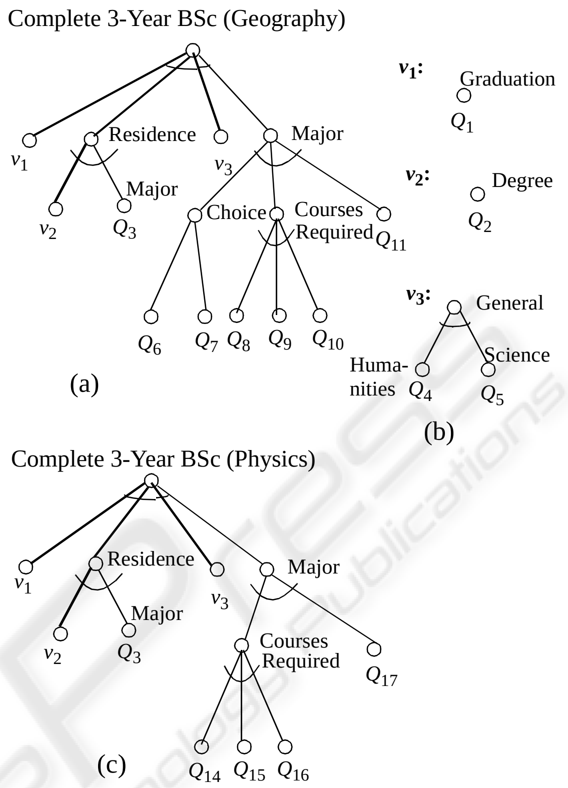


Figure 4: Illustration for SQT fragmentation.

When more than two SQTs are involved, more complicated SQT fragmentation has to be considered as illustrated in Figure 5.

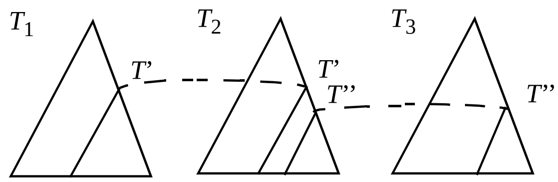
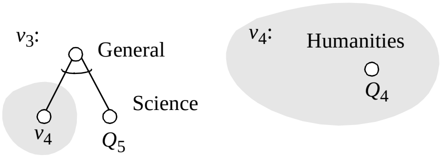


Figure 5: Fragmentation of three SQTs.

In this figure, we show three SQTs: T_1, T_2, and T_3. Among them, T_1 and T_2 share a common subtree *T*'; and T_2 and T_3 share a different common subtree *T*''. Furthermore, *T*'' itself is a subtree of *T*'. In such a case, we will generate five VBSQTs. They are T_1/*T*' (which represents the tree obtained by replacing *T*' with a virtual leaf node in T_1), T_2/*T*', T_3/*T*'', *T*'/*T*'', and *T*''.

As an example, consider a 3-Year BA (English) major where the Humanities requirement is specified, but no specification for the Science requirement. Thus, the subtree representing the Humanities requirement in the SQT for the English major is a proper subtree of the General requirement in the Geography major as illustrated in Figure 6(a), in which v_4 represents the subtree shown in Figure 6(b). Accordingly, the virtual SQT for the English major will be of the form shown in Figure 7.



(a) (b)

Figure 6: Humanities as a fragment of General.

Note that the situation gets more complicated if we allow for some other major that specifies the Science requirement but not the Humanities. In such a case, the General requirement would then have two subtrees common to some different SQTs.

In general, we have the following algorithm to fragment any number of SQTs. The algorithm is followed by an example.

Algorithm *SQT-fragmentation*

1. Let T_1, T_2, ..., T_n be SQT trees;
2. Let Δ^1_{ij}, ..., Δ^k_{ij} be all the subtrees shared by T_i and T_j $(i \neq j)$;
3. Repeat until fragments have been created for all Δ^l_{ij} $(i \neq j)$.

   a) From unmarked subtrees, select and mark Δ^l_{ij} if $\Delta^k_{st} \not\subset \Delta^l_{ij}$ for all Δ^k_{st}, $(i,j \neq s,t)$. Generate a fragment for Δ^l_{ij}. Mark any Δ^w_{uv}, if $\Delta^w_{uv} = \Delta^l_{ij}$.

   b) For each Δ^k_{st}, if $\Delta^k_{st} \supset \Delta^l_{ij}$, and there is no any other Δ^w_{uv} such that $\Delta^k_{st} \supset \Delta^w_{uv} \supset \Delta^l_{ij}$, replace Δ^k_{st} by $\Delta^k_{st} / \Delta^l_{ij}$.

4. Generate VBSQTs: T_1/{all Δ^k_{st}}, ..., T_n/{all Δ^k_{st}}.

To explain the SQT-fragmentation algorithm we consider a more complicated scenario involving Geography Physics, and English, which are considered as T_1, T_2, T_3 respectively in Step 1. Step 2 determines 9 common subtrees as shown below:

For Geography and Physics:

Δ^1_{12} = Graduation subtree

Δ^2_{12} = Degree subtree

Δ^3_{12} = General subtree

For Geography and English:

Δ^1_{13} = Graduation subtree

Δ^2_{13} = Degree subtree

Δ^3_{13} = Humanities subtree

For Physics and English:

Δ^1_{23} = Graduation subtree

Δ^2_{23} = Degree subtree

Δ^3_{23} = Humanities subtree

In Step 3, we first generate 3 fragments for Graduation requirement Δ^1_{12} $(= \Delta^1_{13} = \Delta^1_{23})$, Degree requirement Δ^2_{12} $(= \Delta^2_{13} = \Delta^2_{23})$, and Humanities requirement Δ^3_{13} $(= \Delta^3_{23})$ as these do not contain any identified subtrees (see (a) in Step 3). Then, the fragment for $\Delta^3_{12} / \Delta^3_{13}$ will be created (see (b) in Step 3). Finally, Step 4 generates the following VBSQTs: T_1/ $\{\Delta^1_{12} \cup \Delta^2_{12} \cup \Delta^3_{12}\}$, T_2/$\{\Delta^1_{12} \cup \Delta^2_{12} \cup \Delta^3_{12}\}$, T_3/ $\{\Delta^1_{12} \cup \Delta^2_{12} \cup \Delta^3_{13}\}$.

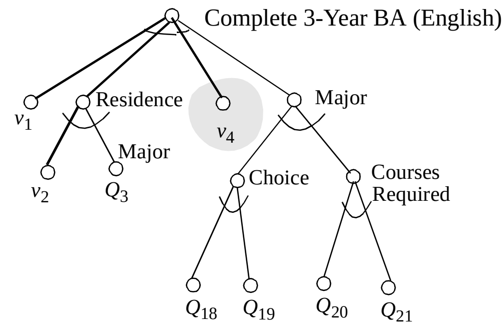


Figure 7: VBSQT for English.

The fragmentation algorithm creates fragments that, for a given set of documents, controls redundancy present in rules by extracting common rules into separate documents. Dividing a collection of documents into sub-documents where those sub-documents are common components is a way of structuring documents into manageable pieces that can be considered separately or in combination.

Figure 8 presents the 3-Year BSc Geography document as presented in Figures 4(a) and 4(b). Note the use of the Xinclude feature of XML (XML.org 2005) to link a pair of documents.

The next section discusses the evaluation of these documents which requires the documents be reassembled in some way.

3.2 Evaluation of Fragmented SQTs

To determine the students who can graduate, we need to evaluate the necessary SQTs for the students in question. If we were to follow the procedure in (McFadyen et al 2005) we would instantiate all SQTs for each degree and every common fragment would be evaluated many times. In contrast, we present another more efficient procedure based on a fragment graph for evaluating graduation status of students.

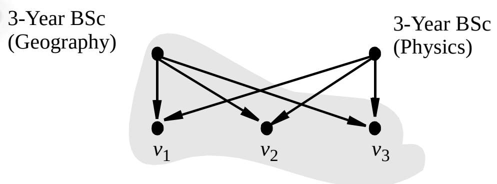
```
<GeographyRule title="Degree Requirement
                      for 3-Year BSc (Geography)"
               combining="AND">
<     Xinclude   href="GraduationRule.xml"  >
<ResidenceRule title = "Residence Requirement"
               combining = "AND" >
<     Xinclude   href="DegreeRule.xml"   >
   <MajorRule   title="Major",  display="minimum 18 credit hours"
                query="... " ...>
   </MajorRule>
</ResidenceRule>
<     Xinclude   href="GeneralRule.xml"      >
<MajorRule>       title="Major Requirement"
                  combining="AND">
   <MinMaxRule
         display="Minimum 30 credit hours, Maximum 48 credit hours"
         query="... " ...>
   </MinMaxRule>"
   <ReqCoursesRule    title="Required Courses"
                      combining="AND">
     <Course
       display="23.202 Intro Geography I" query="..." ...> </Course>
     <Course
       display="23.203 Intro Geography II" query="..." ...> </Course>
     <Course
      display="23.331 Advanced Geography" query="..." ...> </Course>
   </ReqCoursesRule>
   <ChoiceRule    title="Choice"
                  display="23.205 Atmos Sci or 23.206 Earth Sci"
                  combining="OR">
        <Course  query="..."  ...></Course>
        <Course  query="..."  ...></Course>
   </ChoiceRule>
</MajorRule>
</GeographyRule>
```

Figure 8: Requirements document referencing fragments using the XML Xinclude feature.

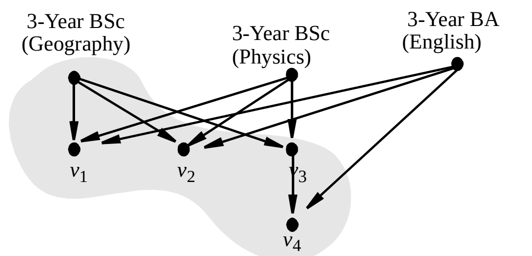
In order to evaluate a fragmented SQT, we construct a directed graph, called a *fragment graph*, in which each node represents a fragment (or say, a VBSQT), and we have an edge from a node $frag_a$ to another node $frag_b$ if $frag_b \subset frag_a$ and there is not any node $frag_c$ such that $frag_b \subset frag_c \subset frag_a$. For instance, the VBSQTs shown in Figures 4, 6 and 7 can be organized into graphs as shown in Figure 9(a) and (b).

To evaluate the status of all students, we evaluate the fragment graph bottom-up. During this process, for each encountered node, we evaluate the VBSQT represented by it and transfer the result obtained to its parents. For example, consider Figure 9(a) for students majoring in Geography and Physics. In a bottom-up fashion, we first evaluate the SQTs represented by v_1, v_2, and v_3. The results are then partitioned according to their majors and sent to the corresponding parent nodes. In the next step, the SQTs represented by the nodes labelled with Geography and Physics are evaluated to find all the students eligible to graduate.

The above process can be improved by using the constants appearing in a query to speed up the computation. For example, if we are considering only the students majoring in Geography, we need to access only part of the graph (marked grey) shown in Figure 9(a) or Figure 9(b).



(a) Fragment graph derived from Figure 4



(b) Fragment graph derived from Figures 4, 6, 7

Figure 9: Fragment graphs.

4 CONCLUSION AND FUTURE WORK

In this paper, we consider a document type that includes requirements and where a user comprehends these requirements as rules to be followed to achieve a certain designation. As a result, we consider each document a single compound rule that may be assembled from many fragments. When such a document (e.g. 3-Year BSc Geography) is evaluated in a certain context (e.g. for a specific student) there will be a value generated for it. For this type of document, fragmented SQTs succinctly represent document content, evaluation and query requirements; a simple tree traversal is required to evaluate or display a document.

Since some requirements may appear many times in different documents, these documents can exhibit a great deal of redundancy. We have introduced an algorithm to fragment a collection of documents, and described an efficient approach for document evaluation where each fragment/VBSQT is evaluated just once, but many results are made available (i.e. for a set of students).

We have developed a prototype system that assembles and displays requirements documents from fragments and determines on request the graduation status for students on a) an individual basis or b) a set-oriented approach for handling many students at one time. The former is useful by an individual student to measure their own progress, and the latter approach is useful in a university setting at say, the end of term, when students should be graduating. The prototype has been constructed using Java, a SAX parser, and student history data stored in a mySQL relational database. Requirements documents are stored as fragments (related via Xinclude) that are independent XML documents. Various functions such as Logical And, Logical Or, Minimum, and Arithmetic Add required for the general synthesized query tree have been implemented.

We are examining algorithms for document evaluation, query optimization, discovery of common subtrees, and other processing models such as the pipe and filter architecture (Albin 2003).

We are examining other situations to apply the query tree approach. We have not used functions that return data in XML format. Such functions can be used to perform an include operation, in the same way that we have used Xinclude. Also, if we allow functions as found in (Abiteboul et al 2002, Abiteboul et al 2003, Bonifati et al 2001) that invoke arbitrary Web Services returning XML then our model allows, as a special case, Active XML documents. We intend to examine other issues related to the processing of these query-based documents including concerns such as “Which major, given my current status, permits me to graduate most quickly?” or “What are the added requirements if I were to do a double major in Geography and Physics, instead of a single major in Geography”?

REFERENCES

Abiteboul, S., Benjelloun, O., Manolescu I., Milo, T., Weber, R., 2002, Active XML: peer-to-peer data and web services integration (demo), in *Proceedings of VLDB.*

Abiteboul, S., Bonifati, A., Cobena, G., Manolescu, I., Milo, T., 2003, Dynamic XML documents with distribution and replication, in *SIGMOD 2003*, June 9-12, 2003, San Diego, CA, USA.

Albin, S. T., *The art of software architecture: design methods and techniques*, Wiley Publishing, 2003, 1st edition, ISBN 0471228869.

Bonifati, A., Ceri, S. and Paraboschi, S., 2001, Active rules for XML: A new paradigm for E-services, in *VLDB Journal*, 10, 39-47.

Business Rules Group, *Defining business rules: What are they really?*, 3rd. edition, July 2000, http://www.businessrulesgroup.org.

McFadyen, R., Chen, Y., Chan, F-Y., 2005, XML-based evaluation of synthesized queries, in *1st International Conference on Web Information Systems and Technologies (WEBIST 2005)*, 24-31, Miami, USA.

Quint, V., Vatton, I., 2004, Techniques for authoring complex XML documents, in *ACM Symposium on Document Engineering (DOCENG 2004)*, Milwaukee, Wisconsin, USA.

Ross, R. G., 1997, *The business rule book*, Business Rule Solutions, Houston, 2nd edition.

Salminen, A., Tompa, F. W., 2001, Requirements for XML document database systems, in *ACM Symposium on Document Engineering (DOCENG 2001)*, Atlanta, Georgia, USA.

XML.org, http://www.w3.org/TR/xinclude/, retrieved June 8, 2005