Tree Pattern Evaluation using SAX

Friso Abcouwer Matthijs van Dorth

May 31, 2014

1 Introduction

In this report we will show how we implemented an algorithm for evaluating tree-pattern queries using SAX. SAX (Simple API for XML) is a way to parse XML document using a stream of data in contrast to DOM (Document Object Model) which is a complete representation of the XML file as a tree.

The parser we created reads a query and an xml file and can display the result of the query in various formats. An example query is given in Listing 1

Listing 1: Query 1

```
1 for $p in //person  [email]
2                     [name/last]
3 return (    $p//email,
4             $p/name/last)
```

Given an XML file such as the one given in the book and in figure 2, the query will return all email and last name nodes of the person nodes that have a email and last name node. The result is a table or an XML file with these values.

Listing 2: people.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<people>
  <person>
    <email>m@home</email>
    <name>
      <first>Mary</first>
      <last>Jones</last>
    </name>
  </person>
  <person>
    <name>
      <first>Bob</first>
      <last>Lang</last>
    </name>
  </person>
  <person>
    <email>@home</email>
    <email>@work</email>
    <name>
      <first>Al</first>
      <last>Hart</last>
    </name>
  </person>
</people>
```

2 The structure of the algorithm

The algorithm that we created has several parts. First there is the PatternNode class that represents a single node in a xml document, such as the node with a name "email". Each node can have a value such as "a@work". When executing a query a TPEStack is created. A TPEStack consists of a PatternNode and a list of child TPEStacks. The TPEStack for Listing 1 can be constructed with the Java code shown in Listing 3.

The TPEStack is used by the StackEval algorithm to create a Match object when a elementname matches the name of the PatternNode. The most important methods of this StackEval algorithm are `startElement`; which is called everytime a opening tag is encountered, `endElement`; which is called everytime a closing tag is found and `characters`; which is called when text nodes are found.

These Match objects keep a record of the TPEStack that was matched against this object and all the child Match object children that still need to get Matched agains a PatternNode.

Once Matches are found, the result is saved into a Result object. This result object has an id, parentId, name, value and depth at which the match was found. This Object is Comparable so that we can sort them on their id. All these results are saved into a ResultList object that is able to print out the results as a table or as XML.

Listing 3: Construction of a TPEStack for Query 1

```
public static TPEStack personStack() {
    PatternNode person = new PatternNode("person");
    PatternNode email = new PatternNode("email");
    PatternNode name = new PatternNode("name");
    PatternNode last = new PatternNode("last");

    TPEStack personStack = new TPEStack(person, null);
    TPEStack nameStack = new TPEStack(name, personStack);
    TPEStack lastStack = new TPEStack(last, personStack);
    TPEStack emailStack = new TPEStack(email, personStack);

    personStack.addChildStack(nameStack);
    personStack.addChildStack(emailStack);
    personStack.addChildStack(nameStack);
    nameStack.addChildStack(lastStack);

    person.addChild(name);
    person.addChild(email);
    name.addChild(last);

    return personStack;
  }
```

Listing 4: Fields of the Result object

```
public class Result implements Comparable<Result>{
  int id;
  int parentId;
  String name;
  String value;
  int depth;
```

3 The Extended Algorithm

3.1 Wildcards

We proceeded with extending the algorithm further by allowing wildcards into the queries. We gave the PatternNode an extra boolean field wildcard which can be set to true or false. In the StackEval algorithm we had to adopt the `startElement` and `endElement` methods to also match when a wildcard was found, this was a minor change. The way the results had to be printed had to be altered though since we needed to retrieve the name of the node in case of printing it as XML or print it with a wildcard symbol *, when printing it in a table.

3.2 Printing the result as XML

The result of the query can be printed out as valid XML. This is done by a method in the ResultList class, that encloses the result in results tags and then prints the opening tags in order and puts

the closing stacks on a stack for later printing. When there are no more results the closing tags are popped from the stack and printed. The depth of each result is used to indent the text.

3.3 Creating a TPEStack from a query

TODO: explain how the Input-Parser works

4 Testing the parser

Using several different XML files we tested the parser with different queries.