A Partition-based Approach for XPath Testing

Claudio de la Riva, José García-Fanjul, Javier Tuya
Department of Computer Science
University of Oviedo
Campus of Viesques, s/n, 33204 (SPAIN)
claudio@uniovi.es, jgfanjul@uniovi.es, tuya@uniovi.es

***Abstract*—The XML language is becoming the preferred means of data interchange and representation in web based applications. Usually, XML data is stored in XML repositories, which can be accessed efficiently using the standard XPath as query language. However, the specific techniques for testing these queries often ignore the functional testing. This work addresses this problem by introducing a technique based on the category-partition method for the systematic design of test input data for an XPath query. The method permits the automatic identification of categories and choices in the XPath and XML Schema implementations and the construction of constraints in order to obtain complete and valid test cases. The technique is illustrated over a practical example.**

Keywords-Software Testing; Category-Partition Method; XML Repositories; XPath

I. INTRODUCTION

The XML language [11] is playing an important role as the standard language for data representation and interchange over Internet. With the growing number of XML documents generated, there is an increasing need in the use of standard XML technologies for the management and the access to XML repositories (*XML data stores*) [10]. Between them, XML Schema [13] is a language for the definition of the XML document structure and data types, and XPath [12] is a query language for elements navigation and selection. In a typical scenario, a web-based application will use one or more XML Schemas to validate the syntactic content and structure of the XML data, and will access the content using XPath queries, perhaps embedded in other languages, such as XQuery, XSLT or Java.

In the development of a software project, testing is a labor-intensive and expensive process which may account for 50 percent of the total project cost. Therefore, more effort is needed in the definition and organization of the testing processes. Between them, the construction of test cases is an important aspect, because it directly affects the effectiveness of the whole process.

However, the specific testing practices in querying XML repositories are scarce, even when important errors can affect critical applications, such as e-commerce applications or web services. The XPath query can specify a complex process which needs to be tested, not only at a syntactic level (the XPath expression), but also at a functional level. Because the semantics of XPath queries are similar to SQL queries, other problems are related to ones found in the database applications [1] [9], such as the lack of a specific methodology for the construction of the test cases and the design of the initial load.

In this work, we present a technique that permits the design of the test input data (XML documents) for functional tests of XPath queries. The test data generation is based on the Category-Partition Method (CPM) [8] by means of the automatic identification of categories, choices and constraints in the XPath query. Because the categories are related to functional characteristics of the query, our approach generates test input data specifically addressed to test functional properties of XPath queries.

The rest of the paper is organized as follows. Section II outlines the related work and the background concepts of XML, XPath and CPM. Section III describes each of the procedures used in the application of CPM to XPath, starting with an algorithm for the identification of categories and choices, then describing the construction of test frames and ending with the procedure to automatically generate test input data as instances of the test frames. Finally, conclusions and future research are presented in Section IV.

II. BACKGROUND

A. Related Work

The approaches that address the problem of testing the access of XML repositories are limited and all recent. In general, they can be classified in two categories.

The first implies the use of tools to analyze and validate XML documents with regard to schemas, for example IBM Scheme Quality Checker, or generate XML documents from a schema, for example XML-DIG.

```
<Univ>
    <Prof id="0">
      <Name>John</Name>
      <Type>A1</Type>
      <Sub>
        <Cod>SWE</Cod>
        <Hours>3</Hours>
      </Sub>
      <Sub>
        <Cod>PRG</Cod>
        <Hours>3</Hours>
      </Sub>
    </Prof>
    <Prof id="1">
      <Name>Mary</Name>
      <Type>A2</Type>
      <Sub>
        <Cod>SWE</Cod>
        <Hours>5</Hours>
      </Sub>
    /Prof>
</Univ>
```

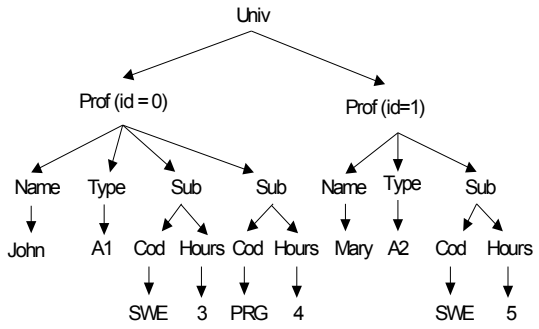

Figure 1. An XML document sample and its hierarchical representation

The second approach is based on the adaptation of classical testing techniques used in other systems. Li and Miller [6] use a mutation-based technique to test the semantics correctness of XML Schema. They define a set of mutation operators, which can be used to detect some faults in the schema. Figueiredo et al [4] define a test process that involves generating XML documents with some modifications with regard to the original. Then they use queries in these documents to validate the schema. The approach is fault-based and, therefore, classes of faults that may be present in the XML Schema are identified. In [7] a partition method based on boundary values to generate XML documents from an XML Schema is presented. The generated XML documents are used to validate web services.

The major difference between the previous approaches and ours is that in the former the testing process is based only in the schema structure. In our work, functional tests are systematically generated from the both XPath and XML Schema implementation.

B. XML

Extensible Markup Language (XML) [11] is a hierarchical markup language used to describe semi-structured data in a way that is independent of its appearance. The XML documents are structured using tags, where the data is placed between `<tag>` and `</tag>`. Fig. 1 shows a well-formed XML document and its hierarchical representation with data about a set of professors and the subjects that they teach in a university.

Sometimes, the XML documents must be valid with respect to schema, which defines the hierarchical structure and data types of the elements. The schema can be defined with a set of grammatical rules using *Document Type Definitions* (DTD) or using XML Schema [13]. In general, XML Schema offers more facilities to define and manage data types.

C. XPath

*XML Path Languag*e (XPath) [12] is a declarative language based on expressions used to navigate and to access the elements of an XML document. An XPath query is formed by one or more location paths. Similar to the file path in the operating systems, a location path specifies the access to elements in the XML document. Analogous to SQL language, XPath queries also can specify filters by means of predicates ("[ ]"). Intuitively, the XPath queries are evaluated from left to right and the result is the set of the elements in the XML documents that satisfy the expression. For example, `/Univ/Prof[Type="A1"]/Name` over the document of Fig. 1 returns the names of the professors with type `A1` (the element `John`).

D. Category-Partition Method

In software testing, the term partition testing encompasses a set of strategies of test case construction that is based on the partition of the input data into a set of classes. Between them, the *Category-Partition Method* (CPM) [8] is frequently used, because it provides a systematic approach to the generation of test cases. CPM consists in the separation of data input into a set of *categories* that represent the main characteristics of the software (for example "Professor Type" in the query of Section II.C). Each category is divided into a set of *choices* that represent the different values which they can take in the category, assuming that every value in a category will produce a similar behavior (for example, "Professor Type=A1" or "Professor Type≠A1"). Then, it is necessary to detect constraints between the different choices in order to construct valid combinations of choices or test frames.

Lastly, the test cases are generated as instances of each test frame.

III. PARTITION TESTING FOR XPATH QUERIES

A. Basic Definitions

In order to provide a mechanism to generate XML documents and capture the most important characteristics, we define a model for XML Schema. The representation is based on the *Regular Tree Grammars* (RTG).

Definition 1. An XML Schema $\mathcal{X}$ is a tuple <E, A, D, N, R, n_0> where:

- E is a finite set of elements
- A is a finite set of attributes
- D is a finite set of data types
- N is a finite set of non-terminal symbols
- R is a finite set of production rules divided into:
  - Non-terminal production, NTR: rules of the form n→e(re), where n∈N, e∈E and re is a regular expression over N
  - Terminal production, TR: rules of the form n→a(d), where n∈N, a∈E or a∈A and d∈D
- n_0∈N, is the initial symbol

Example 1. A schema for the XML document in Fig. 1 could be $\mathcal{X}_U$ = <E, A, D, N, R, n_0>, where:

- E = {Univ, Prof, Name, Type, Sub, Cod, Hours}
- A = {id}
- D = {string, integer}
- N = {n_0, nP, nT, nS, nI, nN, nC, nH}
  - RNT={n_0→Univ(nP*), nP→Prof(nI nN nT nS*), nS→Sub(nC nH)}
  - RT={nI→id(integer), nN→Name(string), nT→Type(string), nC→Cod(string), nH→Hours(integer)}

Definition 2: An XPath query $\mathcal{Q}$ is an expression that can be formed using the following grammatical rules:

```
exp  ::= l | exp op exp | const
l    ::= r | /r | //r
r    ::= p | r/p | r//p
p    ::= . | .. | e ([exp])*
```

The above syntax specifies that an XPath query is formed by location paths (represented by l) and constant values (represented by `const`) with basic logic and relational operators (represented by `op`). The location paths can be absolute or relative. The absolutes start by / (child axis) or // (descendant axis). The relatives (represented by r) consist in a list of steps (denoted by p) connected with / or //. A step can be a self reference (.), a reference to parent element (..) or an expression formed by element names (e) and a sequence of predicates (represented by [exp]*).

Example 2: The XPath query $\mathcal{Q}_U$ `/Univ/Prof[Type="A2"]/Sub[Hours>4]/Cod` over the document of Fig. 1 returns the subject codes of the subjects having hours greater than `4` and taught by a professor of the type `A2`. In the example, this query returns the element `SWE`.

Definition 3: A *path* t^n for the element e_n∈E in a schema $\mathcal{X}$ describes the access from the root element of the $\mathcal{X}$ S to e_n, t^n=/e_1/e_2/.../e_n, e_i∈E. A *query path* t_q for the query $\mathcal{Q}$ is a path in $\mathcal{X}$ for the desired element obviating the predicates. A *predicate path* t_p is a path in $\mathcal{X}$ for a predicate in $\mathcal{Q}$, t_p=/e_1/e_2/.../e_k op v, e_i∈E, op is a relational operator and v is a constant value. Therefore, a query $\mathcal{Q}$ is formed by a query path and zero or more predicate paths.

Example 3: The query $\mathcal{Q}_U$ of the Example 2 is formed by one query path `/Univ/Prof/Sub/Cod` and two predicate paths, one for each predicate in $\mathcal{Q}_U$, `/Univ/Prof/Type="A2"` and `/Univ/Prof/Sub/Hours>4`, respectively.

B. Identification of Categories and Choices

Next, we detail an algorithm that allows identifying the categories and choices for a query $\mathcal{Q}$ and a schema $\mathcal{X}$. The categories are represented between "< >" and the choices between "{}". A choice {e} in the category <C> is denoted as {C:e}.

Step 1. For each predicate path t_p=/e_1/e_2/.../e_k op v:

- Construct the category <C_k> corresponding to the element e_k (if it does not exist).
- Construct at least the choices {C_k:op v}, {C_k:!op v} and {C_k:=∅}[1] (if they do not exist).

[1] It should be noted that more choices could be defined based on the relational operator of the predicate. For example, for the operator ">", the choices {C_k:>v}, {C_k:=v} and {C_k:<v} could be considered

TABLE I. CATEGORIES AND CHOICES FOR $\mathcal{Q}_U$

Path	Step	Category	Choices
/Univ/Prof/Type="A2"	1	<Type>	{Type:≠"A2"} {Type:="A2"} {Type:=∅}
/Univ/Prof/Sub/Hours>4	1	<Hours>	{Hours:>4} {Hours:=4} {Hours:<4}{Hours:=∅}
/Univ/Prof/Sub/Cod	2	<Cod>	{Cod:=∅} {Cod:≠∅}
/Univ/Prof/Sub	4	<#Sub>	{#Sub:=0} {#Sub:>0}
/Univ/Prof	4	<#Prof>	{#Prof: =0} {#Prof:>0}
/Univ	4	<#Univ>	{#Univ:=0} {#Univ:>0}
/Univ	4	<Univ>	{Univ:=∃} {Univ:≠∃}

Step 2. For the query path $t_q=/e_1/e_2/.../e_m$

- If e_m is a leaf element in $\mathcal{X}$, construct the category <C_m> (if it does not exist), with at least the choices {C_m:=∅} and {C_m:≠∅} (if they do not exist).
- If e_m is not a leaf element in $\mathcal{X}$, construct the category <#C_m> (if it does not exist), where the symbol "#" represents "number of", with at least the choices {#C_m:=0} and {#C_m:>0}, (if they do not exist).
- Moreover, if m=1 (e_m is the root element of the schema), construct the category <C_m> with two choices {C_m:=∃} and {C_m:≠∃}.

Step 3. Construct the set T of all paths which are sub-paths of t_q, $T=\{t_q^{m-1}, t_q^{m-2},..., t_q^{1}\}$

Step 4: For each $t_q^i \in T$, perform the Step 2.

Next, we illustrate the algorithm over the schema $\mathcal{X}_U$ and the XPath query $\mathcal{Q}_U$, both described in the Examples 1 and 2 in Section III.A, respectively.

Step 1. For the predicate path `/Univ/Prof/Type="A2"` the category <Type> is constructed with the choices {Type:="A2"}, {Type:≠"A2"} and {Type:=∅}. For the path `/Univ/Prof/Sub/Hours>4`, the choices {Hours:=4}, {Hours:>4}, {Hours:<4} and {Hours:=∅} are constructed.

Step 2. For the query path `/Univ/Prof/Sub/Cod`, the choices {Cod:=∅} and {Cod:≠∅} are constructed.

Step 3. T = {/Univ/Prof/Sub, /Univ/Prof, /Univ}.

Step 4. This step generates the choices {#Sub:=0}, {#Sub:>0},{#Prof:=0}, {#Prof:>0}, {#Univ:=0} and {#Univ:>0}. Moreover, the path `/Univ` identifies the root element of the schema, therefore, the choices {Univ:=∃} and {Univ:≠∃} are also constructed.

Table 1 summarizes the application of the algorithm for the query $\mathcal{Q}_U$. For each category, the path, the step of the algorithm and the generated choices are represented.

C. Test Frames Generation

The next task in CPM consists in the combination of the choices in each category in order to form test frames. Then, the test cases are obtained from the test frames. In absence of constraints between the choices, the number of test cases that could be generated is equivalent to the product of the number of choices in each category (for example, in $\mathcal{Q}_U$ we obtain 384 test frames).

However, many combinations of choices are impossible, because they can not form valid test cases. For example, a constraint in $\mathcal{Q}_U$ is that the choice {Univ:≠∃} can not be combined with any choices in any category.

Given that, implicitly, the schema imposes restrictions in the construction of XML documents, these are used to obtain a set of useful constraints. Intuitively, if there is a category <#C_i> for the element e_i with a choice {#C_i:=0}, this can not be combined with any choice corresponding to elements in an inferior level in the schema. The same is applicable for the choices in the form {C_i:≠∃}. These constraints applied over $\mathcal{Q}_U$ reduce the number of valid test frames to 300.

An important additional characteristic of the test frames is the completeness. According to [1], a test frame F is said to be complete, whenever a single element is selected from every potential choice in F, a test case is formed. From the point of view of the testing, only complete test frames are useful. Moreover, they improve the effectiveness of the test and reduce the number of test cases that must be generated. Thus, the test frame {{Univ:=∃},{#Univ:>0},{#Prof:>0}} is incomplete for $\mathcal{Q}_U$, because we need additional information over the values of the choices in the categories <Type>, <Sub>, <Hours> and <Cod> in order to form a test case for $\mathcal{Q}_U$. Intuitively, a test frame for a query $\mathcal{Q}$ is complete if it contains choices that cover every element in $\mathcal{Q}$, or, it contains choices in the form of {#C:=0} or {C:≠∃}. By considering these constraints, the number of test frames is reduced to 30.

TABLE II. TEST FRAMES FOR $\mathcal{Q}_U$

F_1 =	{ {Univ:≠∃} }
F_2 =	{ {Univ:=∃},{#Univ:=0} }
F_3 =	{ {Univ:=∃},{#Univ:>0},{#Prof:=0} }
F_4 =	{ {Univ:=∃},{#Univ:>0},{#Prof:>0},{Type:=∅},{#Sub:=0} }
F_5 =	{ {Univ:=∃},{#Univ:>0},{#Prof:>0},{Type:="A2"},{#Sub:=0} }
F_6 =	{ {Univ:=∃},{#Univ:>0},{#Prof:>0},{Type:≠"A2"},{#Sub:=0} }
F_7 =	{ {Univ:=∃},{#Univ:>0},{#Prof:>0},{Type:=∅},{#Sub:>0},{Hours:=∅},{Cod: ≠∅} }
F_8 =	{ {Univ:=∃},{#Univ:>0},{#Prof:>0},{Type:="A2"},{#Sub:>0},{Hours:=∅},{Cod:=∅} }
F_9 =	{ {Univ:=∃},{#Univ:>0},{#Prof:>0},{Type:≠"A2"},{#Sub:>0},{Hours: =∅},{Cod:≠∅} }
F_{10} =	{ {Univ:=∃},{#Univ:>0},{#Prof:>0},{Type:=∅},{#Sub:>0},{Hours:=4},{Cod:≠∅} }
F_{11} =	{ {Univ:=∃},{#Univ:>0},{#Prof:>0},{Type:="A2"},{#Sub:>0},{Hours:=4},{Cod:=∅} }
F_{12} =	{ {Univ:=∃},{#Univ:>0},{#Prof:>0},{Type:≠"A2"},{#Sub:>0},{Hours:=4},{Cod:≠∅} }
F_{13} =	{ {Univ:=∃},{#Univ:>0},{#Prof:>0},{Type:=∅},{#Sub:>0},{Hours:>4},{Cod:≠∅} }
F_{14} =	{ {Univ:=∃},{#Univ:>0},{#Prof:>0},{Type:="A2"},{#Sub:>0},{Hours:>4},{Cod:=∅} }
F_{15} =	{ {Univ:=∃},{#Univ:>0},{#Prof:>0},{Type: ≠"A2"},{#Sub:>0},{Hours:>4},{Cod:≠∅} }
F_{16} =	{ {Univ:=∃},{#Univ:>0},{#Prof:>0},{Type:=∅},{#Sub:>0},{Hours:<4},{Cod:=∅} }
F_{17} =	{ {Univ:=∃},{#Univ:>0},{#Prof:>0},{Type:="A2"},{#Sub:>0},{Hours:<4},{Cod:≠∅} }

With the goal of reducing even more the number of test cases, we can impose other constraints related to the fulfillment of a certain coverage criterion [14]. Note that, the test frames generated fulfill the multiple condition criterion, because the complete test frames cover every possible combination of valid choices.

A less strict criterion could be the multiple condition coverage applied only in the choices corresponding to the predicates of the query. For example, in $\mathcal{Q}_U$ only every possible combination of choices in the categories <Type> and <Hours> are considered, but not in the category <Code>. This criterion reduces the number of test frames to 18. Table 2 shows the complete and valid test frames generated in the query $\mathcal{Q}_U$ by applying the multiple condition coverage criteria to the choices related to the predicates of the $\mathcal{Q}_U$.

D. Test Input Data Generation

Using the complete test frames for the query $\mathcal{Q}$, each test case is generated selecting a value from each choice. Given that for a query the number of choices can be greater, the automatic support for the test case generation is an important feature to consider.

In a previous work [3], we presented a technique to generate test input data for XPath queries using test specifications. As test generator, the model checker SPIN [5] is used. The schema $\mathcal{X}$ is represented as a finite state system, where the states are the XML elements of the schema and each production rule defines a transition between the states. Additional insert, modify and delete transitions are provided to the finite state system in order to perform changes in the elements. The test specification is translated to a temporal logic formula that negates the specification. Thus, a counterexample is obtained showing the inconsistency of the negation. For example, if the test specification is "there are professors with type A2", this is translated to a temporal logic formula that states "there is not a professor with type A2". The obtained counterexample shows an instance of $\mathcal{X}$ (XML document), where there are professors with type A2.

The test frames obtained by applying the procedures described in the previous sections represent detailed specifications of the test input data. Therefore, we use the previous approach to automatically generate XML documents as test input data. For each test frame $\{c_1, c_2,...,c_n\}$, where c_i are choices, a temporal logic formula $[]!(c_1 \wedge c_2 \wedge ... \wedge c_n)$ is created, where $[]$ represents the always operator in temporal logic and $\wedge$ is the logic operator AND. Such property asserts that instances in the schema $\mathcal{X}$ fulfilling the conditions represented in each c_i never exist. Fig. 2 shows the test input data (XML document) for the test frame F_{13} in the Table 2 obtained following the previous procedure.

```
<Univ>
    <Prof>
      <Name>P1</Name>
      <Type></Type>
      <Sub>
        <Cod>S1</Cod>
        <Hours>5</Hours>
      </Sub>
    </Prof>
    <Prof>
      <Name>P2</Name>
      <Type></Type>
      <Sub>
        <Cod>S2</Cod>
        <Hours>6</Hours>
      </Sub>
    </Prof>
</Univ>
```

Figure 2. Test input data for the test frame F_{13}

IV. Conclusions and Future Work

In this paper, we have presented a technique based on category-partition that permits the functional test for XPath queries. We have described a systematic procedure for the automatic identification of categories and choices from the XPath expression. Such procedure enables the generation of some constraints to generate valid and complete test frames, and how they are used to construct XML documents as test input data. The process is fully automatic. This aspect must not be considered as a total replacement for the intervention of the engineer in order to complete and/or refine categories, choices or constraints. Even when CPM has been used before in "informal" functional specifications, in this paper we have applied it to obtain functional tests from XPath and XML Schema implementations.

The main future line of work is addressed to reducing the number of generated test frames by means of the definition of adequacy criteria for the query. In this way, the analysis of relations between choices and test frames will be considered. Once completed, we will perform experimentation to measure the effectiveness of the test case generated.

Acknowledgments

This work was funded by the Ministry of Education and Science (SPAIN), National Plan of I+D+i, under the projects IN2TEST (TIN2004-06689-C03-02) and REPRIS (TIN2005-2479-E).

References

[1] D. Chays, Y. Deng, P.G. Frankl, S. Dan, F. Vokolos, E.J. Weyuker: "An AGENDA for testing relational database applications". Software Testing, Verification and Reliability, vol. 14, no. 1, pp. 17-44, 2004.

[2] T.Y. Chen, P. Poon, T.H. Tse. "A choice relation framework for Supporting Category-Partition Test Case Generation". IEEE Transactions on Software Engineering, vol. 29, no. 7, pp. 577-593, 2003.

[3] C. de la Riva, J. Tuya, J. García-Fanjul. "Testing XPath queries using model checking". In Proceedings of Fourth Workshop on System Testing and Validation, 2006, pp. 45-52.

[4] M.C. Figueiredo, S. Vergilio, M. Jino. "A Testing Approach for XML Schemas". In Proceedings of 29th Annual International Computer Software and Applications Conference,2005, vol. 2, pp. 57-62.

[5] G.J. Holzmann. The SPIN model checker: Primer and Reference Manual. Addison-Wesley, Boston, Massachusetts, 2003.

[6] J.B. Li, J. Miller. "Testing the semantics of W3C XML Schema". In Proceedings of 29th Annual International Computer Software and Applications Conference, 2005, vol. 1, pp. 443-448.

[7] J. Offutt, W. Xu. "Generating test cases for web services using data perturbation" ACM SIGSOFT Software Engineering Notes, vol. 29, no. 5, pp. 1-10, 2004.

[8] T.J. Ostrand, M.J. Balcer. "The Category-Partition Method for specifying and generating functional tests". Communications of the ACM, vol. 31. no. 6, pp. 676-686, 1988.

[9] M.J. Suárez-Cabal, J. Tuya: "Using an SQL coverage measurement for testing database applications" ACM SIGSOFT Software Engineering Notes, vol. 29 , no. 6 pp. 253-262, 2004.

[10] A. Vakali, B. Catania, A. Magdalena. "XML data stores: emerging practices". IEEE Internet Computing, vol. 9, no. 2, pp. 62-69, 2005.

[11] World Wide Web Consortium, Extensible markup language (XML) http://www.w3.org/ XML/ (accessed June 2006).

[12] World Wide Web Consortium. XML path language (XPath). http://www.w3.org/ TR/xpath/ (accessed June 2006).

[13] World Wide Web Consortium, XML schema. http://www.w3.org/XML/Schema/ (accessed June 2006).

[14] H. Zhu, P.A.V Hall, J.H.R. May. "Software unit test coverage and adequacy". ACM Computing Surveys, vol. 29, no. 4, pp. 366-427, 1997.