Automatic Test Data Generation for XML Schema-based Partition Testing

Antonia Bertolino, Jinghua Gao, Eda Marchetti, Andrea Polini
Istituto di Scienza e Tecnologie dell’Informazione “A. Faedo” (ISTI-CNR), Pisa
[antonia.bertolino, jinghua.gao, eda.marchetti, andrea.polini]@isti.cnr.it

Abstract

We present the XML-based Partition Testing (XPT) approach for the automatic generation of XML instances from an XML Schema. The approach is inspired by the well-known Category Partition method for black-box testing. The generated instances can be used for inter-operability testing of applications that expect in input conforming XML instances, as well as for other interesting purposes, such as database population, XML Schema benchmarking, web services testing, and so on. The implementation of XPT in a prototype tool called TAXI is described. To limit the number of generated instances, TAXI also incorporates practical strategies for handling element weights and type values.

1. Introduction

As claimed by the AST’07 Call for Papers, notwithstanding the great amount of research effort devoted to automatic test case generation in the past decades, current practice of software testing remains largely manual. With software systems turning out to be "more and more complicated with components developed by different vendors and using different techniques in different programming languages and even run on different platforms", a concern that becomes increasingly crucial is inter-operability.

An important recent innovation for fostering interoperability has been the introduction of the eXtensible Markup Language (XML) [13]. In few years this language has established itself as the de facto standard form for specifying and exchanging data and documents between almost any digital or web applications. Immediately following, the XML Schema [13] has spread up as the notation for formally describing what constitutes an agreed valid XML document within an application domain.

We present an XML-based Partition Testing (XPT) approach, which the proposal is to leverage the great potential of XML Schema in describing input data in open and standard form, to push the automation of test data generation. From a tester’s point of view, in fact, the XML schema expresses the basic rules and constraints on data and parameters that diverse classes of systems and web applications exchange, thus it provides an accurate and formalized representation of the input domain in a format suitable for automated processing, which is clearly a big potential for test automation.

The most traditional approach to software testing [7] is constituted by black-box partition testing, in which a system is tested at its I/O interface by identifying relevant classes of input values, and by systematically choosing some representative test input values for each identified class. The basic assumption behind partition testing is that the input domain can be divided into subdomains, such that, for testing purposes, within each of them the program “behaves the same” (and then for every point within a subdomain the program either succeeds or fails).

Obviously, the degree of automation which is achievable in such a process depends on the formalization level at which the input domain is described: although several researchers have early attempted to exploit formal specifications, the prevalent approach to input domain partitioning remains a semiformal one, as well exemplified by the Category Partition (CP) method [8]. CP provides a stepwise intuitive methodology for the testing of “functional units” from their specifications written in structured, semiformal language.

We claim that the adoption of the XML Schema lends itself quite naturally to the application of partition testing. The subdivision of the input domain into subdomains, according to the basic principle of partition testing, can be done automatically by analyzing the XML Schema elements: from the diverse subdomains identified, the application of partition testing then amounts to the systematic derivation of a set of XML instances.

While several tools exist already which facilitate the derivation of XML instances from an XML Schema with varying degrees of automation, such as [5, 6, 10, 12, 11, 15] (a more comprehensive list of references can be found in [1]), what we propose here is to do such generation in systematic way, borrowing from the well established field of black-box testing. Thus we suppose that a (valid) XML
Schema has been defined, and we exploit that schema for automating the testing of the applications or components that refer to it. To the best of our knowledge there is no other approach that is trying to apply a systematic derivation of XML instances.

However, the pure application of the Category Partition methodology in general leads to the generation of an unmanageable number of instances. Different approaches can be combined with the CP method to overcome this problem. The characteristics of XML Schema seem particularly suited to the adoption of weight-based strategies, as we describe in Section 4.3.

The potential of CP can also be exploited for a different purpose: to automatically derive a set of invalid XML instances useful for verifying the robustness of the tested application. For this we can adapt the same procedures defined for the valid instances derivation, making them generate instances containing a single error at a time. Specifically the instances can be invalid because they contain either a wrong value or a single tag having a structure not compliant with that specified in the XML Schema.

In the remainder of this paper we provide an overview of the Category Partition methodology in Section 2. Then in Section 3 after having shortly introduced XML Schema related concepts we show how it is possible to match it with CP concepts in order to automatically generate meaningful XML instances. Then in Section 4 we introduce a proof-of-concept tool that applies the techniques discussed before. The XPT methodology seems to us particularly promising, therefore in Section 5 we discuss possible scenarios and application domains. Finally in Section 6 we draw some conclusions.

2. Category Partition

The Category Partition (CP) method [8] provides a step-wise intuitive approach to identify the relevant input parameters and environment conditions, as follows:

1. Analyze the specifications and identify the functional units (for instance, according to design decomposition).
2. Partition the functional specifications of a unit into categories: these identify the environment conditions and the parameters that are relevant for testing purposes.
3. Partition the categories into choices: these represent significant values for each category from the tester’s viewpoint.
4. Determine constraints among choices (either properties or special conditions), to prevent the construction of redundant, not meaningful or even contradictory combinations of choices.
5. Derive the test specification: categories, choices and constraints form a Test Specification, suitable for automatic processing. It is not yet a list of test cases, but contains all the necessary information for instantiating them by unfolding the constraints.
6. Derive and evaluate the test frames: from the test specification, a set of test frames is derived by taking every allowable combination of categories, choices and constraints.
7. Generate the test scripts, i.e. the sequences of executable test cases.

3. Mapping XML Schema to Category Partition

A XML Schema describes the structure of an XML document. XML Schema supports data types and namespaces, whereby a namespace is a collection of names identified by URI references. In this section we describe the mapping of the XML Schema in the Category Partition concepts. For the sake of simplicity we used a small example for describing the diverse steps in detail. Figure 1 shows the XML Schema “purchaseForm” (a simple description of a form to order some products via web).

```xml
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="purchaseForm">
    <xsd:complexType>
      <xsd:all>
        <xsd:element name="Address">
          <xsd:complexType>
            <xsd:choice>
              <xsd:element name="USAAddress" type="xsd:string"/>
              <xsd:element name="EUAddress" type="xsd:string"/>
            </xsd:choice>
          </xsd:complexType>
        </xsd:element>
        <xsd:element name="PurchaseOrder" type="xsd:PurchaseOrderType"/>
      </xsd:all>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Figure 1. An example XML Schema.

Specification Analysis: XPT initially analyses the given XML Schema file, in order to simplify the following XML Schema partitioning. This process in XPT is called Preprocessor. This stage can be compared with the specification analysis in CP method. The Preprocessor phase rewrites...
the elements in the XML Schema file with the only exclusion of the `<choice>` element. For instance, the `<all>` elements will be transformed to `<sequence>` by selecting one of the possible sequences of their child elements; the body part of `<group>` element will be copied in elements which refer to them. Note that each transformation has been defined so that the derived final instance are still valid, i.e. they conform to the original XML Schema. The procedure adopted in the Preprocessor phase only makes some transformations that change the syntax but preserve the meaning of the XML Schema elements. As a result, after the Preprocessor phase, the resulting XML Schema includes only `<sequence>`, `<choice>` and simple elements. Figure 2 shows the application of the Preprocessor phase to the Schema of Figure 1. The Preprocessor rewrites the type of “PurchaseOrder” by substituting the reference to “PurchaseOrderType” with full definition of “PurchaseOrderType”. It also transformed “purchaseForm” element from `<all>` into `<sequence>` and provided a random sequence of original `<all>` children elements.

```xml
<xs:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xs:element name="purchaseForm">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="PurchaseOrder"/>
      </xs:complexType>
    </xs:element>
  </xs:schema>
```

**Figure 2. The transformed schema after Preprocessor.**

Functional Units Identification: In this step the XML Schema is divided into a set of subschemas. In the concept of CP method, subschemas can be considered as functional units, which are sub-specifications that can be tested independently. The children of `<choice>` are independent of each other and only one of them can be included in a derived XML instance. Thus the XML Schema is partitioned into distinct sets. Consequently, the functional units can be derived by combining the children of the various `<choice>` elements. As a result, each functional unit is a subschema containing, for each `<choice>` only a child, (chosen among those representing the family of `<choice>`) and is different from the other functional units for the chosen combination of the `<choice>` children. Figure 3 shows the two subschemas derived from Schema of Figure 1. To make the transformation consistent with the Preprocessor phase, in each subschema the element `<choice>` is transformed to `<sequence>` with only one child element: “USAddress” or “EUAddress”.

```xml
<xs:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xs:element name="purchaseForm">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Address"/>
      </xs:complexType>
    </xs:element>
  </xs:schema>
```

**Subschema 1**

```xml
<xs:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xs:element name="purchaseForm">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Address"/>
        <xs:element name="USAddress"/>
      </xs:complexType>
    </xs:element>
  </xs:schema>
```

**Subschema 2**

**Figure 3. The subschemas of “purchaseForm.xsd”**.

Category Partition: The CP categories in XPT correspond to the types of XML elements.

Choice Partition: There are two possibilities: the data that can influence the value of element and the data that affect the structure of the final instances. The former are derived by analyzing the element attributes (such as type, fixed value and default value) and their restrictions (such as minInclusive, minLength and so on). The latter are derived by analyzing the minOccurs and maxOccurs. Occurrence is an important attribute that specifies the number of times an

---

2In the considered Schema there is only a `<choice>` element with two children
element can occur in the parent element. **minOccurs** specifies the minimum occurrence, and **maxOccurs** specifies the maximum occurrence.

**Constraint Determination**: In XPT there are only two types of constraint: “valid” and “invalid”. They are associated with the definition of a set of valid or invalid instance respectively. Thus a “valid” constraint indicates that the values in this **choice** conforms to the specification of the XML Schema. While an “invalid” constraint denotes that the values in the **choice** do not conform to the declaration of XML Schema. Figure 4 shows the XPT category partitioning for the subschema of “purchaseForm”.

![Figure 4. Category partition of “purchaseForm”](image)

### Test Specification Generation:

Test specification in XPT is called **intermediate instance**, which is generated by the combination of the values from each **choice** occurrence. With respect to the CP method, we added a further refinement step to the test specification generation: the application of the Boundary Condition approach to the occurrences of each element. Combining all the possible values of **minOccurs** and **maxOccurs** defined for each element was in fact infeasible. The number of intermediate instances could be extremely large or even infinite. To face this problem we associated specific boundary values taken from the input domains with the occurrences of each element. Considering the subschema1 of Figure 3, two intermediate instances are generated (see Figure 5). In intermediate instance 1 “shipDate” takes the occurrence as 0, which means that in this intermediate instance it will be omitted. In intermediate instance 2 “shipData” takes 1 as occurrence, so it occurs once.

![Figure 5. Intermediate instances from subschema 1.](image)

**Test Cases Generation**: From each intermediate instance a set of final instances can be generated by giving to each element the proper value. The values are selected from the **choices** according to the restrictions expressed in the Schema. Figure 6 shows two final instances that are derived from different intermediate instances, by using the values selected from the database of value.

![Figure 6. Final instances.](image)

---

3By default the **maxOccurs** and **minOccurs** are 1
4. XPT Implementation

The mapping from the CP to the XML Schema Partition Testing has been partially implemented in a proof-of-concept tool called TAXI (Testing by Automatically generated XML Instances) [2]. With respect to what is described in [2], in this section we briefly present the improved architecture adopted to increase the performance of this tool.

TAXI includes four components (corresponding to the four vertical strips in Figure 7):

- **XML Schema Analyzer** (XSA), which implements the XPT method. It expands and preprocesses the XML Schema, prepares the intermediate instance frames and provides a set of final instances.
- **Test Strategy Selector** (TSS), which implements a set of test strategies. It also manages the weight assignment for the elements in the identified functional units.
- **Values Storage** (VS), which manages a database for establishing the boundary values for the occurrences of the elements, and storing the values used for instance generation;
- **User Interface** (UI), which handles user interaction.

Each component includes a set of activities as shown in Figure 7. Specifically the shaded activities are for the weighted test strategies application, the white ones are for XPT methodology, while those with dashed border facilitate the user interaction. In the rest of this section a brief description of each activity is provided.

4.1. User Interaction Activities

**Reading Input XML Schema.** UI reads the input XML Schema and activates two parallel activities: *Weight Assignment* and *Database Population*. This activity creates also a graphical DOM tree, that can be used by users for assigning the weights to the XML Schema elements.

**Database Population.** This activity manages the database that contains the element values used for the final instance derivation. TAXI provides three different options for database population: i) specify the source (for instance a URL) from which the data can be downloaded or the paths to the files containing specific values; ii) manually insert the useful values; iii) recover the values directly from the Schema (for instance by use of the `<enumeration>` restriction).

**Instances Browsing.** This allows the browsing and visualization of the generated instances.

The following sections provide more details about the other activities of TAXI. They represent the core of the Category Partition mapping and the test strategies application.

4.2. The XPT Methodology

**Preprocessing.** This activity implements the Specification Analysis step. With the only exclusion of `<choice>`, Preprocessing rewrites the structure of XML Schema, and takes them into account constructs, like `<all>`, `<simpleType>`, `<complexType>` and so on, and the shared elements, like `<group>`, `<attributeGroup>`, `<ref>`, and type definitions.

**<Choice> Analysis.** This implements the functional unit derivation, the Category Partition, and the Constraints Determination steps. In particular the functional unit derivation implementation, a set of sub-schemas are derived by selecting a different child from each `<choice>` element. In the likely case that more than one `<choice>` elements are present, a combination methodology of `<choice>` children is performed. This ensures that the set of sub-schemas represents all possible structures derivable from `<choice>`. In particular during the Choice Analysis activity the `<choice>` elements in the Schema are modified into `<sequence>`, so that each `<sequence>` only contains one child element. This divides the XML schema into a set of subschemas. The implementation of Category Partition step requires the analysis of the XML Schema definition and the extraction of the useful information. Finally for realizing the Constraints Determination element occurrences and types are analyzed and the constraints are determined from the XML Schema definition.

**Occurrence Analysis.** This activity contributes to the Test Specification Generation implementation. It is focused on the occurrences manipulation and specifically on the defini-
tion of the boundary values for the occurrence of each element. For this last purpose the default values of minOccurs and maxOccurs are fixed to 1. If maxOccurs is associated to “unbounded”, TAXI either asks the user for a suitable value, or uses a fixed value; in the rest of the cases the values minOccurs and maxOccurs are used as boundary values. If invalid structures of instance have to be defined, the activity Occurrence Analysis provides values such that either “occurrence<minOccurs” or “occurrence=maxOccurs”.

Values Management. This activity contributes to the Test case generation implementation. Specifically it selects the value for each element. Values can be generated randomly, or picked up from the value database. For the random generation TAXI takes in consideration the elements’ types, attributes, and constraints specified in the XML Schema definition. For example, if the element type is “integer” with the attribute minInclusive="100", TAXI identifies the following possibilities: select a value from the interval [100, 2147483647] if a valid instance must be derived. Select a value from [-2147483648,100) when a invalid instance have to be generated. If the values are selected from the database the Value Management activity provides values conforming to the constraints determined during the Choice Analysis activity.

Intermediate Instance Derivation. This activity completes the implementation of Test Specification Generation steps. A set of intermediate instance is derived by combining the occurrences of each element. In each intermediate instance the boundary values of the element occurrence are assigned and combined. These values have been defined during the Occurrence Analysis activity and can be valid or invalid occurrence values depend on the generation of valid or invalid instances.

Final Instance Derivation. Final instances are derived from the intermediate instances. During the generation, values of the elements are selected and bestowed on the respective elements. Since the number of intermediate instances could be huge, in the current implementation, TAXI selects only a value for each element. We did not yet implement the combination of values as required in CP application. In case no value is associate to an element in the database the Value Management activity interacts with the user, and asking either to insert the proper values manually or to start a random generation.

4.3. Application of Weights

One of the main issue of the Category Partition methodology is the explosion in the number of instances that it asks for generation. Correspondingly the XPT methodology suffers of the same problem. In TAXI to partially overcome this explosion we implemented a set of weight-based strategies that permit to focus on the most important parts of the Schema. The instances are distributed into the various subschema according to the corresponding weight. In this section we describe briefly the test strategy management implemented in TAXI (see grey activities in Figure 7).

Weight Assignment. Weight is defined in TAXI to denote the importance of elements. The idea underneath the former is that the children of the same <choice> might not have the same importance with regard to instances derivation, and therefore the most important ones from the tester’s point of view should be privileged. TAXI explicitly requires the user to annotate each children of a <choice> element with a value (called the weight), belonging to the [0,1] interval, representing its relative “importance” with respect to the other children of the same <choice> (clearly the sum of the weights associated to all the children of the same <choice> element must be equal to 1). The default assignment is that all children have the same weight. For each option TAXI then derives the so-called “final weight”, as the product of the weights of all nodes on the complete path from the root to this node. Such value expresses roughly the ideas of how risky is that child and how much effort should be put into the derivation of instances containing it.

Strategy Selection. In order to make the generation more flexible, TAXI provides three test strategies to formulate the result of derived instance set, which are:

Applying TAXI with a fixed number of instances, that could be in practice the case in which a finite set of test cases must be derived. The instances are generated from all subschemas according to their final weights.

Applying TAXI with a fixed functional coverage, when a certain percentage of functional test coverage (e.g. 80%) is established as an exit criterion for instances generation (and then consequently for testing). The subschemas are selected from the highest to the lightest by their weight, until the sum of weights is equal or greater than the coverage. Then all instances from these subschemas are generated.

Applying TAXI with a fixed functional coverage and number of instances, the above mentioned strategies are combined. TAXI first selects the proper substructures useful for reaching a certain percentage of functional coverage and then derives the proper number of instances considering the selected subschemas weights.

On one side the application of weight-based strategies might decrease the overall effectiveness of test cases, but on the other side, by these strategies the test effort can focus on the most critical parts of the schema, and omit the instances affecting those parts that have marginal impact on the overall system behavior, or the functionalities that are considered as trustable.

5. Exploring Possible Application Domains

The proposed approach for the automated and systematic
derivation of instances from an XML Schema specification can be useful in differing situations:
* for validating database management systems. In fact, several tools and techniques have been previously proposed to exploit the potential of XML Schema for benchmarking techniques [14], [9]. TAXI would help to automatically generate valid XML instances for populating database in a systematic way: this could be extremely useful for evaluating the performance and the quality of the associated management systems;
* for testing the inter-operability between applications and for enabling the correct interactions among the interfaces used by remote components in distributed systems. The generation of valid and invalid instances enables the automated testing of I/O behavior;
* for verifying the proper communication protocols between web-services using SOAP-based interaction between services can be reconduted to the corresponding XML Schemas.

Indeed some initial experiments have been already conducted in a restricted domain such as XML instances transformation [3]. Tools, in form of stylesheets, are developed in order to transform XML instances, conforming to a given into XML Schema, to other XML instances to be conforming to other XML Schema. Testing would be useful in general to verify that the stylesheets define correct transformation rules. The availability of an XML Schema for the derived XML instances provides a kind of oracle, thus solving one of the greatest hurdle in the application of the CP method and then of TAXI. In [3] we have shown how the whole testing process of XML stylesheets can be completely automated. We listed some of the possible application domains for which there are clear opportunities to the application of TAXI. Nevertheless the wide usage of XML technologies in many different domains will certainly open the way to extend to application of the discussed methodology to many other fields.

6. Conclusions and Future Work

In this paper we illustrated the XML-based Partition Testing XPT approach, which provides a systematic and automatic method to generate a representative set of XML instances from an XML Schema. So far no proposal has really succeeded in pushing the widespread adoption of automated black box testing as it would deserve. We think that the widespread acceptance of XML, and its pragmatic flavor, could be the winning instruments to do so. A proof-of-concept tool TAXI has been implemented based on the XPT methodology. Additionally TAXI also implements strategies for the use of weights and value management, that can help make the instances suites tractable and meaningful.

As discussed in the paper we propose our approach as a viable solution towards widespread adoption of different application domains, such as database population, benchmark generation and web-service application testing.

Over the last years the problem of automatic instances generation from a XML Schema has been investigated and several proposals have been presented. However these approaches only implement random or ad hoc generation, so they do not attempt to cover all possible elements combinations of the schema. Our research focused instead on adopting a systematic criterion for instances generation which can automatically apply a XML based black-box testing method and provide more accurate and mindful XML instances generation.

In the future we intend to continue the development the TAXI tool (in particular the generation of invalid instances is still limited) and try to combine XPT with combinatorial approaches, such as the AETG methodology [4], to reduce the generated instances further.

References