XMODEL-BASED TESTING OF XSLT APPLICATIONS

Antonia Bertolino, Jinghua Gao, Eda Marchetti, Andrea Polini
Istituto di Scienza e Tecnologie della Informazione "Alessandro Faedo"
Consiglio Nazionale delle Ricerche
Via Moruzzi, 1 – 56124 Pisa, Italy
{*antonia.bertolino, jinghua.gao, eda.marchetti,andrea.polini*}*@isti.cnr.it*

Keywords: Automatic Tests Generation, Category Partition, XSLT Testing

Abstract: Model-based testing is nowadays the emerging paradigm for software testing in many domains. In the Web arena XML Schema is becoming the technology of reference to describe data structure and applications input domains. The proposed tool (TAXI - Testing by Automatically generated XML Instances) exploits such a model to automatically derive correct XML instances applying the well-known Category-partition methodology. In this paper we introduce an improvement of TAXI to test XSLT Stylesheets. Indeed, with XSLT Stylesheets increasingly getting larger and more complex, their correctness becomes a crucial factor for software quality and hence we believe that they need careful validation. Two different case studies illustrate the approach to the validation of XML to XML and XML to XHTML transformations.

1 INTRODUCTION

Generally speaking a model is a schematic representation of an idea or information including all the characteristics and properties useful for a specific purpose in a precise context. Considering in particular the testing phase, models are necessary and extremely important. However the weakest point of models is that many times models exist only in the human minds or rely on domain knowledge of people involved in the diverse activities.

Model based testing is nowadays one of the widespread techniques adopted for validating and verifying applications properties. Even if one of the main arguments against model-based testing is the effort necessary for constructing or maintaining the models and/or annotating them properly for making easier the testing phase. Many industrial realities relies on this technique for guiding, managing and executing the testing phase.

In the context of web applications a fundamental step for the widespread adoption of models has been the adoption of standard formats and open data specifications, and most notably the recent introduction of the eXtensible Markup Language (XML) (W3CXML, 1996). The XML Schema (W3CXMLSchema, 1998) has then spread up as the notation for formally describing what constitutes an agreed valid XML document within a certain application domain.

Thus XML Schemas represent explicit and formal models of data in which all the type constraints and possible data combinations are represented. From this descends naturally the possibility of introducing a Model Based Testing over XML, i.e., readapting commonly adopted black-box testing techniques (for instance partition testing approaches (Bertolino et al., 2005)), by exploiting the XML Schema characteristics. Translated in terms of testing an XML Schema means having a clear representation of the input (output) domain which make easier the construction of test values and the partitioning of them.

Usually the identification of the proper input partitions strongly depends on the skill of the tester and the background knowledge s/he has about the values domain. In XModel-Based Testing the XML Schema provides an automatic subdivision of the input domain into subdomains, according to the basic principle of partition testing. (la Riva et al., 2006). From the diverse subdomains identified, the application of partition testing amounts to the *systematic derivation* of a set of XML instances. Systematic generation of

XML instances, differently from a random based approach, clearly has important consequences on the effectiveness of the generated test suite permitting to derive meaningful statistics on the kind of instances generated, and then on the covered features.

In this perspective, a proof-of-concept tool, called TAXI (Testing by Automatically generated XML Instances) for the systematic generation of XML instances (Bertolino et al., 2006) according to the XML-based Partition Testing (XPT) method has been previously developed. It takes in input an XML Schema and automatically generates a set of XML instances, by implementing several criteria and heuristics, so that it can be used as test data for the black box testing of a component, whose expected input conforms to the taken schema.

The contribution of this paper is exploit the potential of the tool TAXI for testing the XSLT Transformation tool. If from one said there are explicit models of the data domain, i.e. XML Schemas, correspondingly there is the more increasing needs of mechanisms for transforming one format into another. Technologies developed for such purpose require the definition of documents called Stylesheets. These define how data described within a specific input document should be transformed and formatted into the target document. With reference to the context of Web applications certainly the most successful specification for such purpose is that of the eXtensible Stylesheet Language Transformation (XSLT) (W3CXSLT, 1999).

However writing down explicitly the rules that allow specific transformations sometimes is a quite complex work especially when the transformation output should adhere to precise XML format, such as Schemas. In such cases the possibility of making errors in the stylesheet could be considerable and in parallel the difficulty in identifying them could have a hight increase. As a consequence, due to the importance and the criticality of these documents, the necessity of methods for automatically testing the correctness of the stylesheets and quickly identifying their source of errors arise.

This paper wants to face exactly this last need, and provides, by means of TAXI, a testing framework for automatically and systematically generate test cases for verifying the XSLT Stylesheets correctness. We therefore assume that the the input and the output of the XSLT Stylesheets are and have to be, respectively, conform to given XML Schemas. Thus we use TAXI, for automatically generate the set of test cases to be run with the tested XSLT Stylesheet, and with and a XML Validatior for automatically check the correctness of the transformations.

In the rest of the paper we present an overview of the related works (Sec. 2) and a brief description of the TAXI tool (Sec. 3). Then we describe the application use of TAXI for testing the XSLT Transformation Processor (Sec. 4) and the application of the integrated testing framework to some case studies (Sec. 5). Conclusions are briefly drawn in (Sec. 6).

2 RELATED WORKS

Nowaday XSLT is used widely in web, document composition, electronic and print publishing applications. Increasingly, in some cases, XSLT Stylesheet becomes the core of document creation and processing pipelines. XSLT Stylesheets are getting larger and more complex, and containing the information of business logic. The correctness of XSLT Stylesheets then becomes an extremely important factor that can affect the software quality.

There are a lot of tools that already exist for XSLT testing. Most of them do unit testing to XSLT Stylesheets. Such as UTF-X (Radajewski and Daniel, 2006), which is an extension to the JUnit Java unit testing framework and provides functionality for unit testing XSLT stylesheets and strongly supports the test-first-design principle; tennison-tests (tennison, 2006), which allows user to write unit-tests in XML, and exercising XSLT from Ant; and Alster (Alster, 2006), which is an XSLT unit testing framework, allows the creation and "execution" of XSLT Stylesheets containing specially marked "test templates".

The main weakness of these tools is in the selection of test cases, which is mainly based on manual setting. Since XSLT becomes more and more complex, the costs of test cases creation is rising rapidly.

3 TAXI

In this section we briefly describe the XPT methodology. For space limitations, we assume a basic knowledge of XML and XML Schema, and in the text we will refer to the elements of XML Schema using this `format`.

The XPT methodology is largely inspired to the well-known Category Partition (CP) technique (Ostrand and Balcer, 1988), which provides a stepwise intuitive approach to identify the relevant input parameters and environment conditions and combine their significant values into an effective test suite. While the original CP technique relied on a specification expressed in natural or semiformal language, the XML Schema today provides an accurate representation of

the input domain and lends itself quite naturally to the automated application of CP.

The XPT methodology consists of four different components (see Fig. 1):

- XML Schema Analyzer (XSA), which preprocesses the XML Schema applying some initial transformations (see details below), and successively manages the final instance generation phase.
- Test Strategy Selector (TSS), which implements diverse test strategies useful for selecting the elements of the XML Schema to be used for instance generation. This allows the distribution of final instances covering the most critical parts of a Schema.
- Values Storage (VS), which is in charge of storing the values used for instances generation and managing a common database;
- User Interface (UI), which implements and controls the interaction between the user and the tool.

These four components work in agreement, according to the scheme shown in Figure 1. Following this schema, in the rest of this section we briefly describe the main processing steps of TAXI.

Input XML Schema. The UI first reads the input XML Schema and then starts two further activities that proceed in parallel: *Weight Assignment* and *Database Population* in charge of the TSS and VS components, respectively.

Weight Assignment. 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.

Database Population. It provides the tester with the possibility of memorizing set of interesting values for each of the element considered in the XML Schema. In particular the activity provides three different options: 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) leave the user with the possibility of manually inserting the useful values; iii) recover the values directly from the Schema in those cases a list is provided (for instance by use of the `enumeration` restriction).

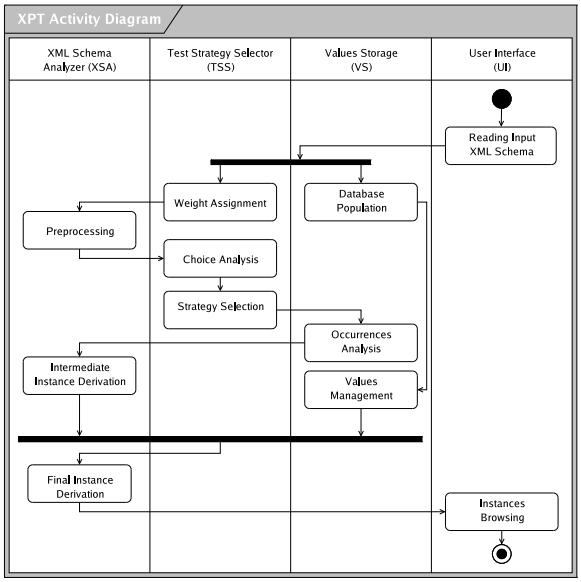

Figure 1: XPT main activities

Preprocessing. The XPT methodology proceeds then with a *Preprocessing* step in which the other XML Schema constructs different from `choice`, like `all`, `simpleType`, `ComplexType` and so on, and the shared elements, like `group`, `attributeGroup`, `ref`, and `type`, are analyzed and manipulated. Considering, for instance, the `all` elements, one of the possible sequences of their children elements is randomly chosen and used for generating instances; considering the `group` elements, their body is copied in each element which refers to them.

Choice Analysis. In case of several `choice`s within one schema, as many subschemas as the number of the possible combinations of the children of the `choice` nodes are produced. Once this operation is completed, the set of the partial subtree weights is normalized so that the sum of the subtree weights over the entire set of substructure is equal to 1.

Strategy Selection. The subtree weights are then used in the *Strategy Selection* activity. Specifically three different test strategies are implemented:

- *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.
- *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).
- *Applying TAXI with a fixed functional coverage and number of instances*: the above mentioned strategies are combined. XPT first selects the proper substructures useful for reaching a certain

percentage of functional coverage and then considers the subtree weights of these selected subschemas and normalizes them so that their sum is still equal to 1.

Occurrences Analysis. The XPT methodology proceeds with the activity called *Occurrences Analysis*, pertaining to the VS components, which analyzes the occurrences declared for each element in the subschema and, applying a boundary condition strategy, derives the border values (minOccurrences and maxOccurrences) to be considered for the final instances generation.

Intermediate Instance Derivation. The XSA components then constructs a set of intermediate instances structures, each one derived by the combinations of the border elements occurrences. These are roughly modified tree representations of the various sub-schemas in which special tags and instructions are introduced to make the final instance derivation easier.

Value Management. The latter step is done in parallel with the *Value Management* activity. According to the number of occurrences, established for each element during the *Intermediate Instance Derivation* phase, the VS component randomly retrieves from the database the required number of values. If the stored values are not sufficient, the *Value Management* activity interacts with the user, asking either to insert the proper values or to start a random generation. During this activity, predefined values available in the Schema and various constraints (for instance `facets`) are also considered.

Final Instance Derivation. According to the test strategy selected and using the values provided by the *Value Management* step, the *Final Instance Derivation* phase produces and stores the final set of instances, which then corresponds to the final test suite, in a suitable directory.

Test Instances Packaging and Browsing. This is the last activity of the XPT methodology. It allows the browsing and visualization of the generated instances by the user.

4 APPLYING XMODEL-BASED TESTING

This section describes how XML Model based testing can be applied to the verification of Stylesheets defining transformations over XML instances into another.

Specifically we focus on the XSLT Tranformation Processors which are the interpreters of the language that, taken in input the starting XML file, read the stylesheet defined using XSLT and apply the rules to generate the target document. Figure 2 shows the process of transforming one XML instance into another.

Many stylesheets have been defined by developer just for deriving HTML documents retrieving data stored into XML documents. Nevertheless this is not the case with other application tools that use XML data formats to interoperate. In such cases the document must strictly adhere to an agreed format (XML Schema) and any minimal divergence from the specified schema can affect interoperability. The basic assumption in this context is that the input and the output instance documents shown in Figure 2 are and have to be, respectively, conform to given XML Schemas. Our insight is that the XModel-Based Testing approach can be helpful here providing an effective way to test a defined transformation.

Figure 3 provides an overview of how model based testing can be applied in this setting to verify the correctness of a stylesheet. The basic idea is to use the TAXI tool, introduced in Section 3, to derive a testing framework (called "TAXI XSLT Testing" - TXT) that permits, in a largely automatic way, to verify the correctness of a transformation stylesheet.

First steps of the resulting process is the automatic and systematic generation of a large number of instances, derived by TAXI guaranteeing the conformance to the XML Schema. In fact TAXI derives instances with many different characteristics that permits to test the specified transformation in many different contexts. Next step of the process foresees than the transformation of each single instance into a target document. At this point the assumption that the target instances must conform to an XML Schema gives us a model of the oracle that can be used to check the correctness of the instances generated by the stylesheet transformation.

The XML based check carried on in the described framework provides a necessary conditionjust of syntactical nature for the correctness of the stylesheet. This means that the stylesheet produces an instance that will be judged correct with respect to the XML schema but however it could be different from the intentions of developer. Nevertheless having a completely automatic testing process that use already developed models (such as XML Schema) largely augment the verification power of stylesheet transformations. The verification power of the main process shown in Figure 3 could be easily augmented if the final check can use other mechanisms not only based on the XML Schema conformance check. For instance the developer could develop a specification using a rule based language (e.g. Schematron (Schema-

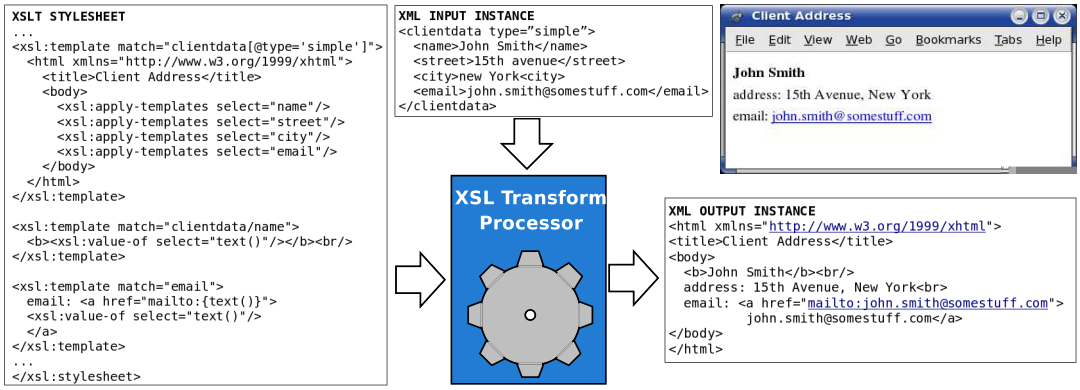

Figure 2: An example of tranformation using an XSLT stylesheet

tron, 2006)) that permit to specify more content related checks. At the same time in some cases the final check could be conducted not based on the XML Schema but using a validator developed by someone else. Next section will provide two different examples using different kind of check for the XML output instances.

5 CASE STUDY

In this section we will show two case studies. One case study is to test the XML to XML transformation. In this case both of these XML files must be conformed to the specific reference XML schemas. Another one is to test an XSLT stylesheet that does the transformation from XML file to XHTML. The XML file must conform to an XML schema, and the XHTML must be well-formed.

5.1 XML to XML Transformation Testing

The first case study considers a customer management system require, which is the transformation from XML into another XML using XSLT stylesheet. The customer management system has two versions. In old version, the data of customers are stored as the structure that defined by the XML schema that shown in Fig 4 (source XML schema [1]). The customer data are defined as the attribute of the element `customer`.

But in the new version of the management system, the structure of customer information is changed into the schema shown in Fig 4 (target XML schema). In the new structure `customer` has only one attribute,

[1]For space limitation we do not show the source schemas, instead we create simplified versions of them

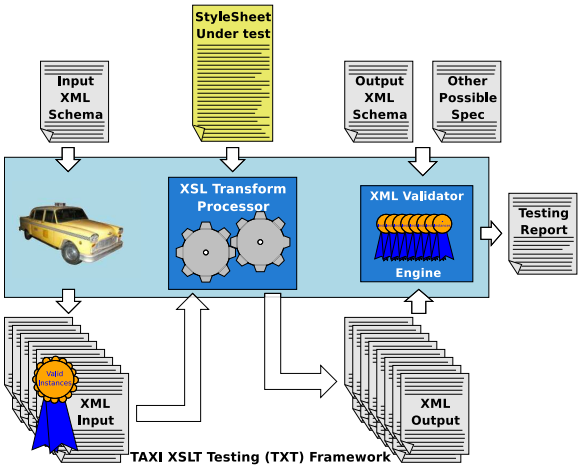

Figure 3: The XSLT Model-based testing framework

which is `CustomerID`, other data are stored as it's child elements.

An XSLT Stylesheet is created to transform all customer records from the old structure to the new one, which is shown in Fig 4 (XSLT stylesheet). Since the XML schema defines only the structure of the file the XML instances from the schema can have different values, and structures. And that XSLT stylesheet must transform all possible XML instances that can be generated from the input XML schema properly. That is why we use TAXI to test XSLT template, and ensuring the correctness of the transformation.

Using TAXI to test the XML to XML transformation, all of the source XML schema, target XML schema and XSLT stylesheet are required taking as input. Then choosing "XSLT Testing" from the TAXI interface. The testing is done on-the-fly and automatically. When an XML instance derived from the source XML schema, TAXI then transforms it according to the XSLT immediately. And the transformed XML file (we call it target file) will be validated by

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="customers">
<xs:complexType>
<xs:sequence>
<xs:element name="customer" minOccurs="1" maxOccurs="unbounded">
<xs:complexType>
<xs:attribute name="CustomerID" type="xs:string"/>
<xs:attribute name="CompanyName" type="xs:string"/>
<xs:attribute name="ContanctName" type="xs:string"/>
<xs:attribute name="ContanctTitle"  type="xs:string"/>
<xs:attribute name="Address" type="xs:string"/>
<xs:attribute name="City" type="xs:string"/>
<xs:attribute name="PostalCode" type="xs:integer"/>
<xs:attribute name="Country" type="xs:string"/>
<xs:attribute name="Phone" type="xs:integer"/>
<xs:attribute name="Fax" type="xs:integer"/>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>
```

Source XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="customers">
<xs:complexType>
<xs:sequence>
<xs:element name="customer" minOccurs="1" maxOccurs="unbounded">
<xs:complexType>
<xs:sequence>
<xs:element name="CompanyName" type="xs:string"/>
<xs:element name="ContactName" type="xs:string"/>
<xs:element name="ContactTitle" type="xs:string"/>
<xs:element name="Address" type="xs:string"/>
<xs:element name="City" type="xs:string"/>
<xs:element name="PostalCode" type="xs:integer"/>
<xs:element name="Country" type="xs:string"/>
<xs:element name="Phone" type="xs:integer"/>
<xs:element name="Fax" type="xs:integer"/>
</xs:sequence>
<xs:attribute name="CustomerID" type="xs:string"/>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>
```

Target XML Schema

```
<xsl:stylesheet version="1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="xml"/>
 <xsl:template match="/">
  <customers>
   <xsl:for-each select="/customers/customer">
    <xsl:element name="{name()}">
     <xsl:for-each select="@*">
      <xsl:element name="{name()}">
       <xsl:value-of select="."/>
      </xsl:element>
     </xsl:for-each>
    </xsl:element>
   </xsl:for-each>
  </customers>
 </xsl:template>
</xsl:stylesheet>
```

XSLT Stylesheet

Figure 4: Case study1 - Input files.

the target XML schema automatically. After that the result of validation will be recorded into the test report file. Testing will not stop until the XML generation is complete. User can find the test report, source XML files and the target XML files in the folder *TAXI_instances* on disk C.

In the first case study, there are two XML instances that generated from the source XML schema, and two target XML files, which transformed by XSLT stylesheet. Due to space limitations, in Fig 5 only one source XML file and one target file are presented.

The test report that is shown in Fig 6 gives the outcome of the testing. The result is “testing failed”. Errors in the test report are derived from XML validator, which point out the invalid places in the target XML files. According the test report, we can easily to discover that the fault is leading by the transformation of `CustomerID`. XSLT should keep it as attribute, but transform it as child element. So the target XML files

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<customers>
<customer CustomerID="pefjvhyh" CompanyName="pzwewmlt"
  ContanctTitle="eptoljwg" ContanctName="otlrwghj"
  Address="gphnheyk" City="pznugbym" PostalCode="80353152"
  Country="fgsgsr" Phone="269445419" Fax="1189064027"/>

<customer CustomerID="dfsgf" CompanyName="fxsft"
  ContanctTitle="adfhh" ContanctName="hfsg"
  Address="sdfhj" City="jhggruy" PostalCode="456536"
  Country="gsgshht" Phone="56353" Fax="5636356"/>

<customer CustomerID="rsgg" CompanyName="sfgsg"
  ContanctTitle="adfag" ContanctName="fhdd"
  Address="fadfa" City="afhjm" PostalCode="4577"
  Country="fadfyt" Phone="24575369" Fax="254787"/>
</customers>
```

Source XML file form TAXI

```
<?xml version="1.0" encoding="UTF-8"?>
<customers>
 <customer>
  <CustomerID>pefjvhyh</CustomerID>
  <CompanyName>pzwewmlt</CompanyName>
  <ContanctTitle>eptoljwg</ContanctTitle>
  <ContanctName>otlrwghj</ContanctName>
  <Address>gphnheyk</Address>
  <City>pznugbym</City>
  <PostalCode>80353152</PostalCode>
  <Country>fgsgsr</Country>
  <Phone>269445419</Phone>
  <Fax>1189064027</Fax>
 </customer>

 <customer>
  ............
 </customer>

 <customer>
  ............
</customers>
```

Target XML file transformed by XSLT

Figure 5: Case study1 - Source and target XML files

are not conform to the target XML schema. According to the analyse of the test report, the bug in XSLT can be revealed easily.

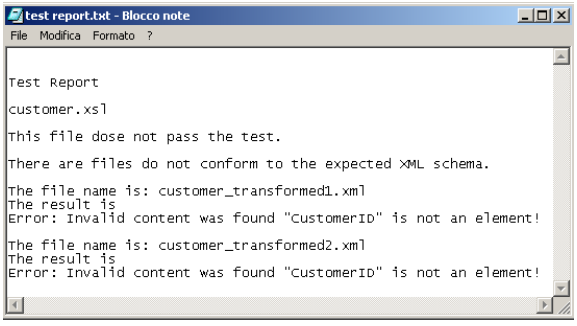

Figure 6: Case study1 - Test report

5.2 XML to XHTML Transformation Testing

In many domains the main application of XSLT stylesheets is transforming XML to XHTML, which will be the second case study. The source XML schema is shown in Fig 7 (source XML schema), which defines the data structure of CD category. The information is shown by XML form, which is very difficult to be understood by people who is not an expert of XML. The motivation of transformation is giving those data a nice outlook and showing them in the clear and neat form. The XSLT Stylesheet is shown in Fig 7 (XSLT stylesheet).

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:complexType name="cdType">
   <xs:sequence>
      <xs:element name="title" type="xs:string"/>
      <xs:element name="artist" type="xs:string"/>
      <xs:element name="country" type="xs:string"/>
      <xs:element name="company" type="xs:string"/>
      <xs:element name="price" type="xs:decimal"/>
      <xs:element name="year" type="xs:gYear"/>
   </xs:sequence>
 </xs:complexType>
 <xs:element name="category">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="cd" type="cdType" maxOccurs="unbounded"/>
      </xs:sequence>
   </xs:complexType>
 </xs:element>
</xs:schema>
```

Source XML Schema

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <html>
  <body>
    <h2>My CD Collection</h2>
    <table border="1">
    <tr bgcolor="#9acd32">
      <th align="left">Title</th>
      <th align="left">Artist</th>
    </tr>
    <xsl:for-each select="catalog/cd">
    <tr>
      <td><xsl:value-of select="title"/></td>
      <td><xsl:value-of select="artist"/></td>
    </tr>
    </xsl:for-each>
    </table>
  </body>
  </html>
</xsl:template>
</xsl:stylesheet>
```

XSLT Stylesheet

Figure 7: Case study2 - Input files

Currently the testing of the transformation between XML and XHTML is different with the transformation between XML and XML. The target XML schema of XML to XHMTL is optional. If it is provided by user, the XHTML will validated by that schema. Otherwise, like in the case study, the target schema is omitted, the validation will use only the specification of XHTML to check if the XHTML is well-formed. However that gives more flexibility to the testing.

The test report is shown in Fig 8, all transformed XHTML files are pass the validation, so the XSLT stylesheet pass the testing.

6 CONCLUSIONS AND FUTURE WORK

In this paper we presented an integrated testing environment useful for validating the correctness of XSLT Transformation processes. For this, according to the emerging Model-based testing paradigm, we exploit the TAXI tool, which uses the information provided by the XML Schema for automatically deriving test cases, which are here XML instances. TAXI has been now integrated with an XML Validator, which verifies the correctness of the transformation against the target XML Schema.

Of course the performance of the proposed testing environment strictly relies on the use of XML Schemas both for the source and the target XML files involved in the transformation. These structures increase considerably the interoperabiliy and efficiency of the Web applications based on XML. Therefore our claim is that the XML Schemas, considered as "models" of the input/output domain, can also be used for automatic testing, and in our view these will represent an interesting new field in the model based testing community. For this we introduced the concept of XModel-Based testing, and showed how this applies quite naturally to the validation of XSLT transformations.

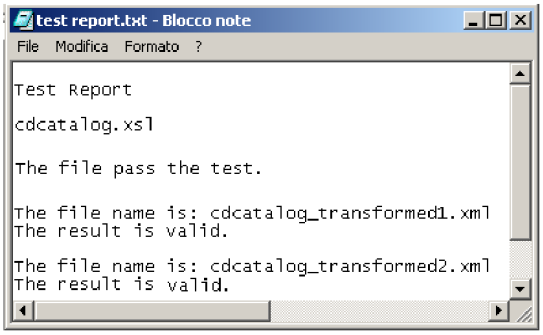

Figure 8: Case study2 - Test report

REFERENCES

Alster (2006). Alster-XSLT Unit Testing Framework. http://alster.sourceforge.net/.

Bertolino, A., Gao, J., Marchetti, E., and Polini, A. (2005). Partition testing from xml schema. Technical report, ISTI-CNR. Technical Report n. ISTI-2005-TR-45.

Bertolino, A., Gao, J., Marchetti, E., and Polini, A. (2006). Systematic generation of xml instances to test complex software applications. In *Rapid Integration in Software Engineering, (RISE 2006)*. to appear in LNCS. Geneve, Switzerland.

la Riva, C. D., Garca-Fanjul, J., and Tuya, J. (2006). A partition-based approach for xpath testing. Tahiti, French Polynesia. IARIA, IEEE Computer Society.

Ostrand, T. and Balcer, M. (1988). The category-partition method for specifying and generating functional tests. *Communications of ACM*, 31(6).

Radajewski, J. and Daniel, A. (2006). Utf-x: Unit testing framework - xslt. http://utf-x.sourceforge.net/.

Schematron (2006). Schematron. http://www.schematron.com/.

tennison (2006). tennison. http://tennison-tests.sourceforge.net/index.html.

W3CXML (1996). W3C XML specification. http://www.w3.org/XML/.

W3CXMLSchema (1998). W3C XMLSchema specification. http://www.w3.org/XML/Schema.

W3CXSLT (1999). XSL Transformations (XSLT). http://www.w3.org/TR/xslt.