International Journal of Computer Science
and Engineering (IJCSE)
ISSN(P): 2278-9960; ISSN(E): 2278-9979
Vol. 4, Issue 2, Mar 2015, 9-18
© IASET

International Academy of Science,
Engineering and Technology
Connecting Researchers; Nurturing Innovations
IASET

JAVA BASED XSD, XML AND WSDL VALIDATION TOOL

MARMIK PANDYA, SIDDHARTH SHAH & ANUP PANDEY

Auto & Auro BU, L&T Infotech Ltd., Mumbai, Maharashtra, India

ABSTRACT

With XML (eXtensible Markup Language) becoming the standard for representing and exchanging data on the Internet, the problem of validation of XML data when updated has attracte more and more attentions. The traditional validation processes used to take hours with high risk of having to miss many checks and hence error prone.

Our paper presents an automated validation tool, based on Java, which provides an easier and faster way to validate the XSD and WSDL files based on the company standards of Larsen & Toubro Infotech.

KEYWORDS: XML, WSDL, XSD, SOAP, MVC, JAXB, SAXParser

INTRODUCTION

In the modern era of communication, web services have taken an upper hand over all other means of communication between any two electronic devices over the internet. Web services being a popular and convenient way of communication follow SOAP standards for their implementation worldwide, making it platform independent, generic and hence reusable.

To be able to reuse such a varied functionality as much as possible there is an urge of a standards to be followed across the applications or parties involved in communication.

Standardization means having more rules and hence adding more work in the implementation of the web services. Thus in order to NOT to divert the aim of having more reusable and structured web service on the Internet we try to automate the task of standardization in the first step.

This tool validates XSD and WSDL as per the predefined standards. The tool is developed in Java which takes a set of WSDL and XSD files used for designing a web service as an input; performs read (unmarshal) and write (marshal) operations in the XML file; validate tags (automatically as well as manually) and generates a report of the validation performed which can be saved at any location in your computer in Excel spreadsheet format.

SYSTEM DESIGN

As per the scope of the project, the system follows this architecture:

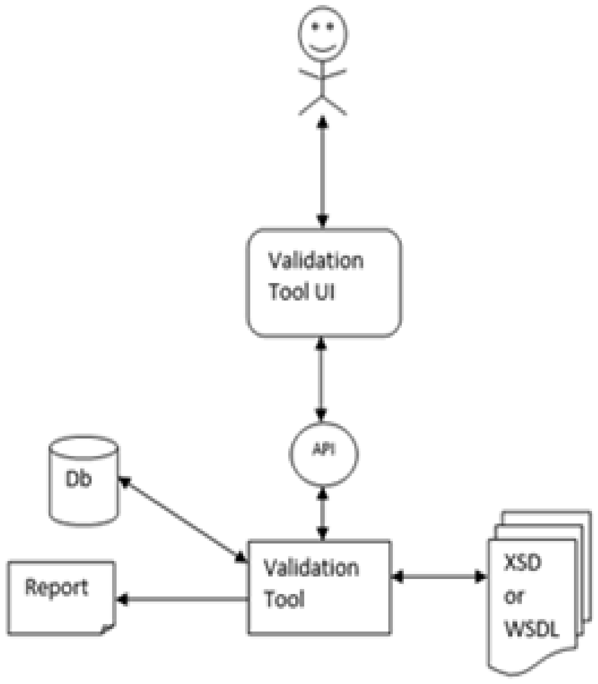


Figure 1: Architectural Representation of the System

Use-Case View Model

It depicts the use-case or scenarios from the project architecture:

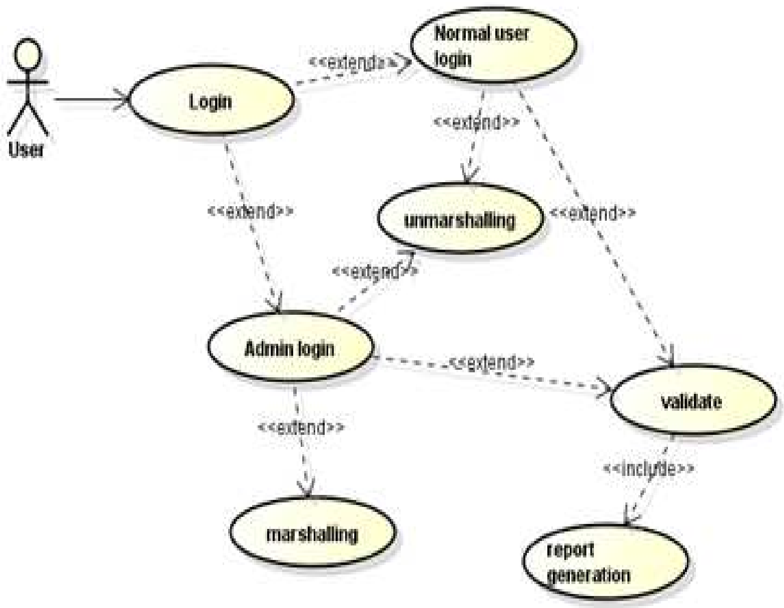


Figure 2: Use-Case Diagram

Sequence Diagram

The following sequence diagram shows detailed view of how reading (unmarshalling) and writing (marshalling) takes place:

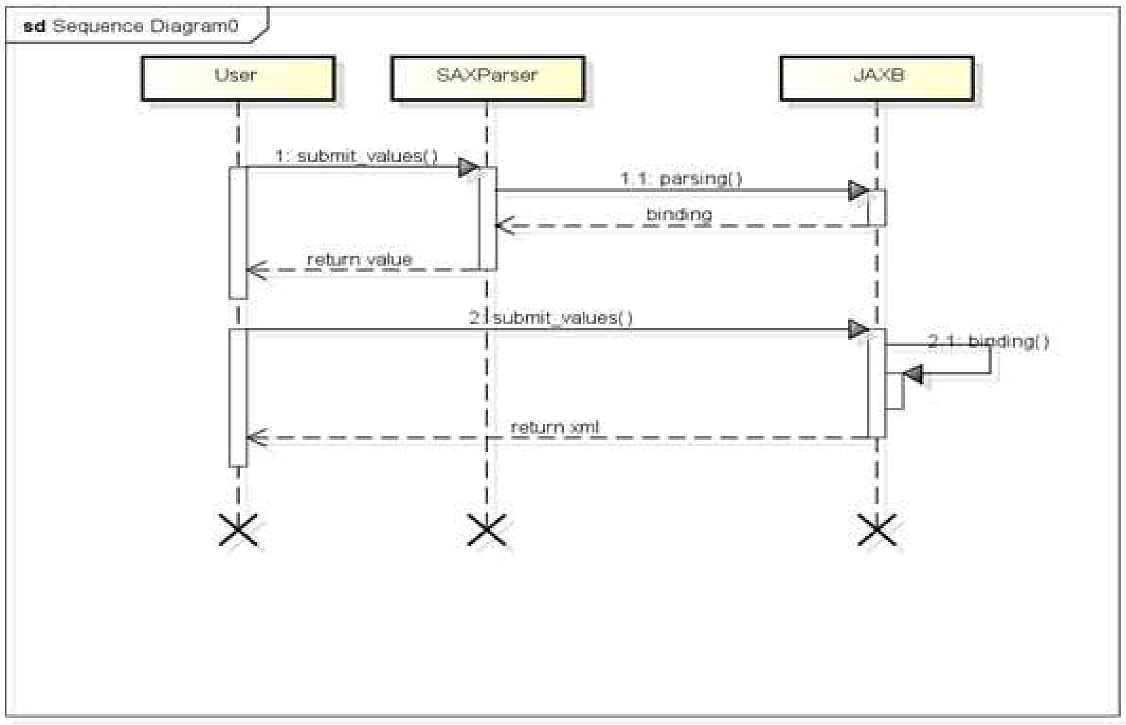


Figure 3: Sequence Diagram

IMPLEMENTATION

The code implementation of the tool follows Object Oriented Approach which makes the components of the tool are modular as well as reusable. Since, the tool performs three different activities on the set of files, the implementation of each were independent.

Implementation of this project involves creating the database for the login, creating the User Interface for the system, writing code for processing user requests & generating and printing report after validating. The project has been designed in such a way so as to create a simple, user-friendly GUI, although it involves complex backend programming.

MARSHALLING

Marshalling is done here in order to follow 'write' operations. It is done so as to write tag values into an XML file. This process converts Java objects into XML. The tag values submitted by user through the tool UI is parsed into Java objects using SAXParser, and JAXB writes them off into an XML file.

UNMARSHALLING

Unmarshalling is done here in order to follow 'read' operations. It is done so as to read tag values from an XML file. This process converts XML values into Java objects. The tool retrieves the tag values of the tag names submitted by user through the tool UI. The tag names submitted is parsed into Java objects first using SAXParser which is binded into XML object using JAXB. A code for searching that XML object in the referenced XML file has been implemented. As soon as the queried XML object is found, the tool retreieves the tag attributes or values of the tag names searched by the user. It then rebinds the XML objects into Java objects using JAXB and is returned to the user using display.

Validation means to check if the generated XSD file of a web service follows proper standards or standard. Most used standards for web service is SOAP. The standards has been set as per the requirements of the Auto & Auro BU of L&T InfoTech

Let’s see a sample XSD file, which is referenced for validation for the demo version of the tool:

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema id="ConnectedAdministrationService"

targetNamespace="http://xmlns.sample.com/fleetmanagement/schema/ConnectedAdministrationService/v1"elementForm
Default="qualified"      xmlns="http://xmlns.sample.com/fleetmanagement/schema/ConnectedAdministrationService/v1"
      xmlns:tns="http://xmlns.sample.com/fleetmanagement/schema/ConnectedAdministrationService/v1"
      xmlns:xs="http://www.w3.org/2001/XMLSchema"
      attributeFormDefault="qualified" version="1.0">

 <xs:element name="GetStaffRequest">
  <xs:complexType>
   <xs:sequence>
    <xs:element minOccurs="1" name="staffRef" type="tns:StaffRef"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>

 <xs:element name="GetStaffResponse">
  <xs:complexType>
   <xs:sequence>
    <xs:element minOccurs="0" name="staff" type="tns:Staff" />
   </xs:sequence>
  </xs:complexType>
 </xs:element>
 <xs:element name="PingRequest">
  <xs:complexType>
```

```
   <xs:sequence>
    <xs:element minOccurs="1" name="executionProcess" type="tns:PingType" />
    <xs:element minOccurs="1" nillable="true" name="responseTimeMax" type="xs:int" />
   </xs:sequence>
  </xs:complexType>
 </xs:element>

 <xs:element name="PingResponse">
  <xs:complexType>
   <xs:sequence>
    <xs:element minOccurs="1" name="resultCode" type="tns:PingResultCode" />
    <xs:element minOccurs="1" nillable="true" name="resultDescription" type="xs:string" />
    <xs:element minOccurs="1" nillable="true" name="responseTimeActual" type="xs:int" />
    <xs:element minOccurs="1" name="serviceVersion" type="xs:string" />
   </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

The validation checks of this XSD file has been done as per the functionalities stated in the Functional Requirements (*see Section 3.2*). The validation report follows a color coding for the understanding of the user: ‘Green’ for correct validation, ‘Red’ for incorrect validation, and ‘Yellow’ for warning. The report generated after the validation of this XSD file.

Functional Requirements

The functional requirements for the validation of the XSD are:

- **Namespace Structure**

  The given Namespace structure is http://xmlns.sample.com/[domain]/[type]/[object]/v[version]

- **Functionalities**

  checkNamespaceStructure for ConnectedAdministrationService_v1.0.wsdl

  checkNamespaceStructure for fleetmanagement_connectedadministrationservice_v1.0.xsd

Namespace Prefix Character Check

Rules for prefix: - Always use lower case - a-z, for first letter - a-z, 0-9, (period) and - (hyphens) for the rest

- **Functionalities**

checkNamespacePrefixStructure for ConnectedAdministrationService_v1.0.wsdl

checkNamespaceStructure for fleetmanagement_connectedadministrationservice_v1.0.xsd

Namespace Pefix Length Check

The recommendation is to use between 3 and 5 letters for the prefix

- **Functionalities**

checkNamespacePrefixLength for ConnectedAdministrationService_v1.0.wsdl

checkNamespacePrefixLength for fleetmanagement_connectedadministrationservice_v1.0.xsd

Namespace Version Check

The version of the object, a letter v followed by a number. The number is the major version. e.g. v2

- **Functionalities**

checkNamespaceVersion for ConnectedAdministrationService_v1.0.wsdl

checkNamespaceVersion for fleetmanagement_connectedadministrationservice_v1.0.xsd

Type Check Namespace

Type can be one of the following: -Schema for XML -Contract for WSDL

- **Functionalities**

checkNamespaceType for ConnectedAdministrationService_v1.0.wsdl

checkNamespaceType for fleetmanagement_connectedadministrationservice_v1.0.xsd

Target Namespace Check

Target namespace is there when there are user defined schemas. UseTarget namespace attribute with following attributes: elementFormDefault, attributeFormDefault

- **Functionalities**

checkTargetNamespace for fleetmanagement_connectedadministrationservice_v1.0.xsd

Some functions are to be manually checked. They are:

- Backward Compatibility: Contracts should be backward compatible
- Attachments: Attachments Profile Version 1.0 is the foundation when using attachments in messages
- URL Check: URL should not resolve in an actual network address and also no document should be present on that address

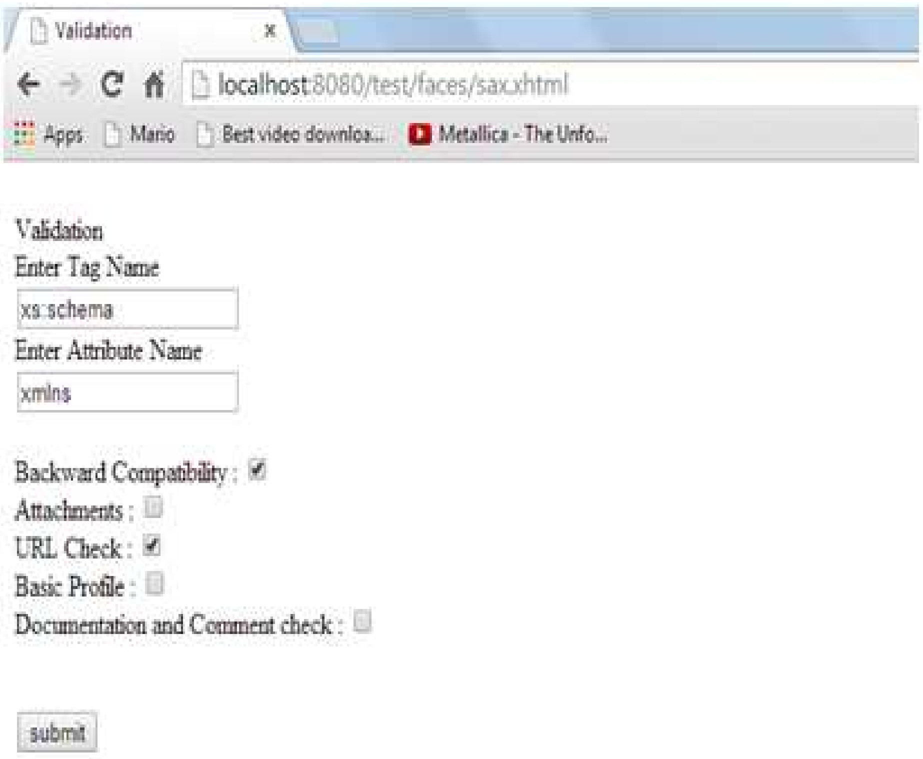
- Basic Profile: Service must be defined according to WS-I Basic Profile 1.1
- Documentation and CommentCheck: Check if xsd:documentation or wsdl:documentation elements are used for comments

RESULTS

After successful implementation of the validation tool, the results have been observed and recorded for possible further future work.

Validation

The inputs for validation are the tag name and attribute name to be validated automatically. For the validation checks that cannot be performed automatically, checkboxes has been provided for the user to check them manually.



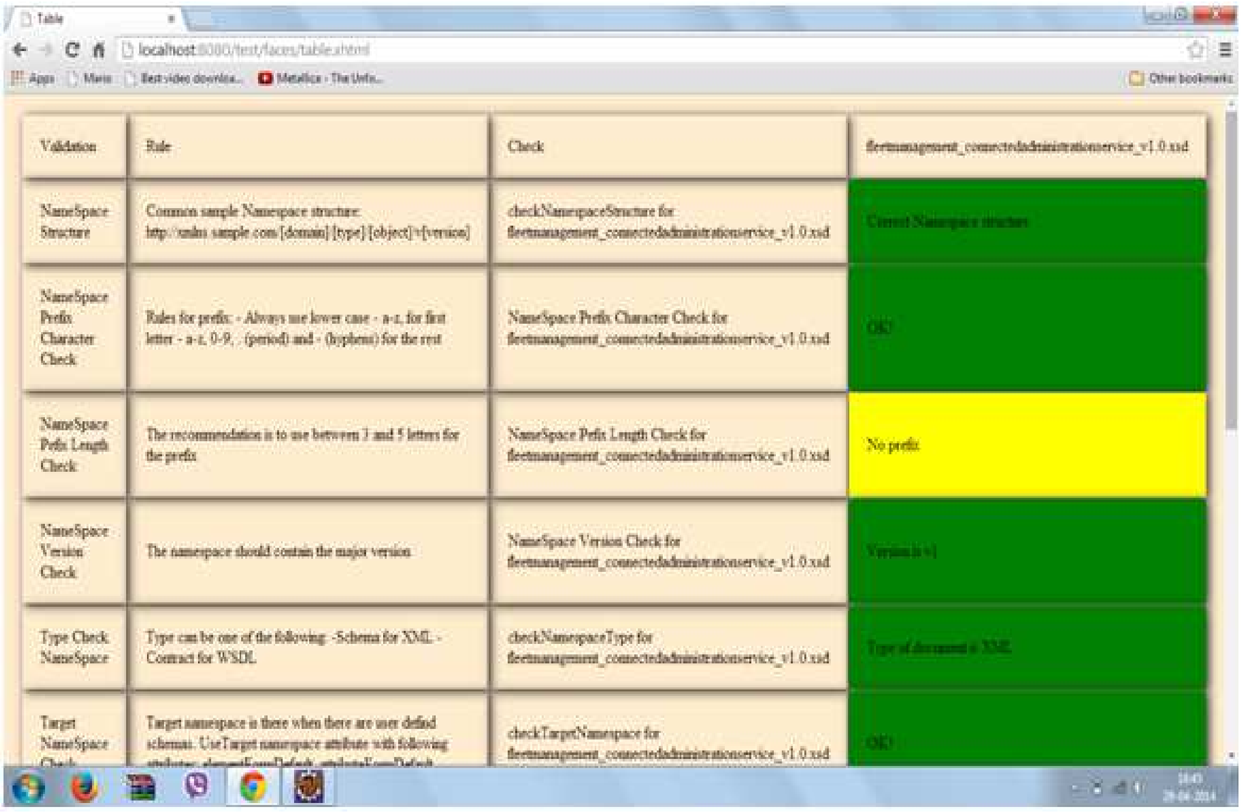
Validation
Enter Tag Name
xs:schema
Enter Attribute Name
xmlns

Backward Compatibility : [✓]
Attachments : [ ]
URL Check : [✓]
Basic Profile : [ ]
Documentation and Comment check : [ ]

submit

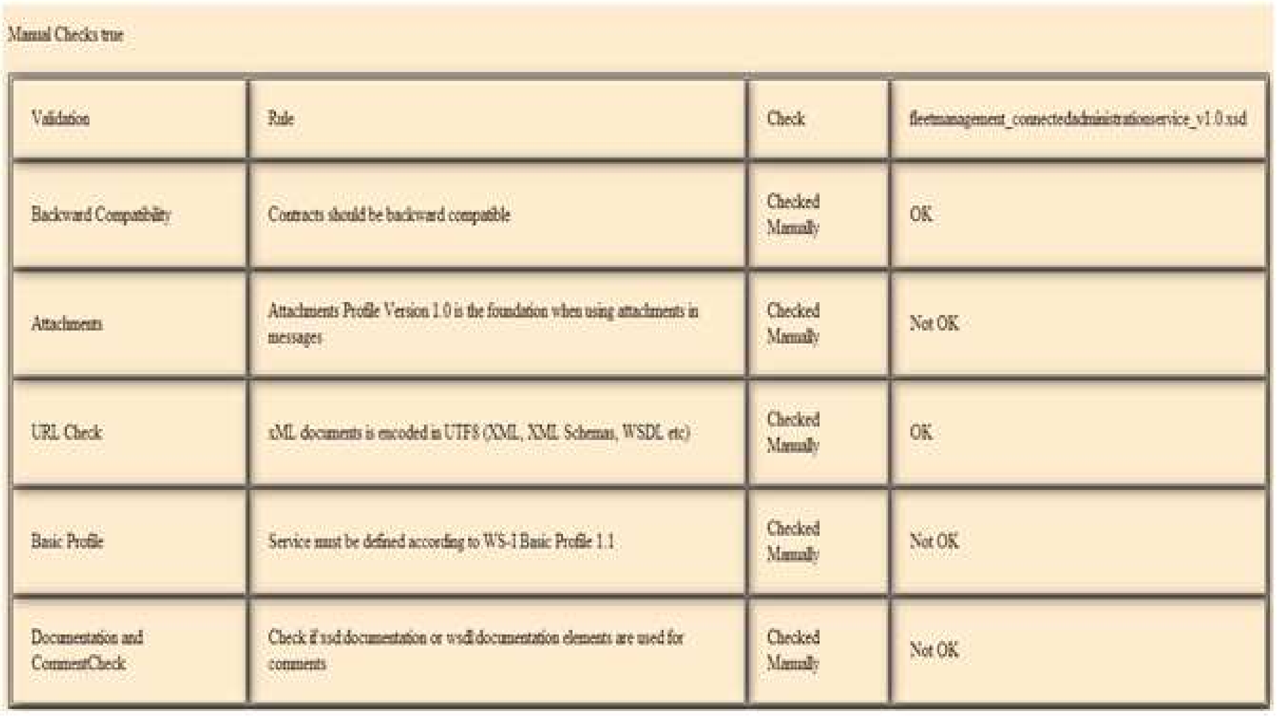
Figure 4: Validation Input

After the values has been submitted, the tool generates a report made in CSS3 which show the validation checks graphically.



Validation	Rule	Check	fleetmanagement_connectedadministrationservice_v1.0.xsd
NameSpace Structure	Common sample Namespace structure: http://xmlns.sample.com/[domain]/[type]/[object]/v[version]	checkNamespaceStructure for fleetmanagement_connectedadministrationservice_v1.0.xsd	Correct Namespace structure
NameSpace Prefix Character Check	Rules for prefix: - Always use lower case - a-z, for first letter - a-z, 0-9, . (period) and - (hyphens) for the rest	NameSpace Prefix Character Check for fleetmanagement_connectedadministrationservice_v1.0.xsd	OK!
NameSpace Prefix Length Check	The recommendation is to use between 3 and 5 letters for the prefix	NameSpace Prefix Length Check for fleetmanagement_connectedadministrationservice_v1.0.xsd	No prefix
NameSpace Version Check	The namespace should contain the major version	NameSpace Version Check for fleetmanagement_connectedadministrationservice_v1.0.xsd	Version is v1
Type Check NameSpace	Type can be one of the following: -Schema for XML -Contract for WSDL	checkNamespaceType for fleetmanagement_connectedadministrationservice_v1.0.xsd	Type of document is XML
Target NameSpace Check	Target namespace is there when there are user defind schemas. UseTarget namespace attribute with following attributes: elementFormDefault, attributeFormDefault	checkTargetNamespace for fleetmanagement_connectedadministrationservice_v1.0.xsd	OK!

Figure 5: Validation Report as Generated in Browser



Manual Checks true

Validation	Rule	Check	fleetmanagement_connectedadministrationservice_v1.0.xsd
Backward Compatibility	Contracts should be backward compatible	Checked Manually	OK
Attachments	Attachments Profile Version 1.0 is the foundation when using attachments in messages	Checked Manually	Not OK
URL Check	xML documents is encoded in UTF8 (XML, XML Schemas, WSDL etc)	Checked Manually	OK
Basic Profile	Service must be defined according to WS-I Basic Profile 1.1	Checked Manually	Not OK
Documentation and CommentCheck	Check if xsd documentation or wsdl documentation elements are used for comments	Checked Manually	Not OK

Figure 6: Validation Report of the Manual Checks

FUTURE SCOPE

The tool can be developed for larger scope working. Some of which can be suggested as:

- To add ‘browse’ functionality to add files from the system in .zip format rather than through referenced path.
- To be able to host the tool on live server.
- To add AGUD (Add-Get-Update-Delete) functionalities.
- To be able to download and save the report in more portable and compatible formats, like .pdf.

CONCLUSIONS

Several mechanisms and XML APIs can be used to parse and validate XSD file like Validate using internal DTD, Validate using internal XSD, Validate using external Schema. The mechanism can mostly stay the same for these different APIs.

We have developed a system that validates a XSD file thus removing all the error in the file and making the web service run smoothly as well as saving a lot of time, as doing the same activity takes hours with high risk of having to miss many checks and hence error prone.

REFERENCES

1. G. Sudha Sadasivam “Middleware Enterprise and Integration Technologies”, Wiley India edition
2. http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html
3. http://docs.oracle.com/javaee/5/tutorial/doc/bnapk.html
4. http://docs.oracle.com/javaee/5/tutorial/doc/bnapl.html
5. http://docs.oracle.com/javaee/5/tutorial/doc/bnaty.html
6. http://www.jsftutorials.net/