Identification and Analysis of Coupling Factors in Service Interface

Ananya Preeti Padma [1] **Prakash Chandra Dash** [2]**Edi Laxmi**[3]

[1]Asst. Professor, Einstein Academy of Technology & Management, Bhubaneswar
[2]Asst.Professor, Einstein Academy of Technology & Management, Bhubaneswar
[3]Student, Einstein Academy of Technology & Management, Bhubaneswar

Abstract—**Coupling is the utmost important design characteristics of Service Oriented Architecture. In the services, service interface is the only binding between service consumer and service provider. The service interface has to be designed first. During the design it has to hold the design characteristics of service oriented architecture. Coupling is directly related with the reusability and performance of the service. Therefore the measurement of degree of coupling at the design stage is extremely important. As our focus is to propose metrics for evaluating coupling in service interface, this paper identifies and analyses the structural factors that causes coupling.**

Keyword- Coupling, coupling factors in service interface, SOA, coupling in XML schema

I. Introduction

Coupling is an elementary and important concept in software and software development process. In the context of object oriented programming Coupling is the degree to which one class knows about another class. The continuous change in technology and needs of user causes continuous change in the software also. It will increase the maintenance cost and effort. In order to reduce this effort and cost, the software is divided in to number of modules which are as independent as possible. When one module know about the another module, then there is a dependency between them. The existent of dependency between modules, components or services is called as coupling. In Service Oriented Architecture (SOA), the principle characteristic is loose coupling. The dependency exists between the services affects the flexibility of the system [1]. Loose coupling reduces the impact of change in the system when new functions are inserted into the system [2].

Service Oriented Architecture always relies on the fundamental design principles. To comply with the coupling, the service and consumer should not interact directly. They should communicate through service messages. These messages are the realizations of the service interface. In SOA, the web services are communicated with each other through their interfaces. The interface of a web service can be defined by the combination of WSDL and XML schema [5][6].

The design techniques that can be applied to the service interface highly determine the coupling of the service. In order to measure the degree of coupling in service interface designs the structural elements which causes coupling has to be identified. The service interface is 819with xml schema. In the service interface different types of schema structures and elements are used to define the data[3]. When one element refers the other element directly or indirectly causes dependency between them. As the number of references increases, the degree of coupling will be increased. This paper identifies and analyses the structural elements of xml schema which are used to design service and also causes coupling.

II. Global Elements And Global Attributes

Elements and attributes are defined in xml schema as either global or local. A global element or attribute is the one defined as an immediate child of the <schema>. If an element or attribute is nested in another component, it is said to be local. Global elements or global attribute will be reused from the target schema as well as from other schema documents. What will happen if all the elements or attributes are declared as global? If all the elements or attributes are declared as global, the number of attribute or element references increased. It will also increase the complexity and thereby increasing of coupling of the schema [4]. Sometimes except one element, all elements may be declared as local elements (Example-1). Here all the elements are bundled together as a single element. No element is depending on another element. The changes in these elements have less impact. The complexity is also less. Since there is no dependency we can say there is no coupling.

```
<xsd:element name="Book">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element name="Title" type="xsd:string"/>
            <xsd: element name="Author" type="xsd:string"/>
            <xsd: attribute "language" type="xsd:string"  />
         </xsd:sequence>
      </xsd:complexType>
   </element>
```

Local Element and attribute Declaration (Example-1)

In other case, to break up the system into more number of small components, all the elements may be declared as global and they are assembled by means of references (Example-2). Since the elements are referenced, if there is change in one element the impact will be high. The complexity is also high. There is a dependency between the elements and hence coupling is also high.

```
<xsd: element name="Title" type="xsd: string"/>
<xsd:element name="Author" type="xsd:string"/>
<xsd: attribute name="language" type="xsd:string"  />

   <xsd:element name="Book">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="Title"/>
            <xsd: element ref="Author"/>
            <xsd:attribute ref="language" />
         </xsd:sequence>
      </xsd:complexType>
   </xsd: element>
```

Global Element and Attribute declaration (Example-2)

III. Type References

The element type declaration can be specified with the “type” attribute. The XML schema types can be classified into two types namely Simple Type and Complex Type[7]. The user can independently define these types. A simple type is a type that only contains text data that can be used with element and attribute declarations. The simple type is used when a restriction is placed on an embedded simple type to create a new user type which is explained in section IV-B of this paper. The complex type is a type which contains one or more elements or attributes (Example-3).

```
<xs:element name="Name" type="xs:string" />
<xs: element name="Department" type="xs: string" />

<xs: complexType name="NameType">
  <xs:sequence maxOccurs="unbounded">
    <xs:element ref="Name" />
    <xs:element ref="Department" />
  </xs:sequence>
</xs:complexType>

<xs: element name= “student”  type= “NameType”/>
<xs: element name= “teacher”  type= “NameType”/>
```

(Example-3)

In the above example NameType is a complex type declaration which contains two children Name and Department. Once a complex type is defined any number of elements can be defined of that type. Here the types are referred. In the example the elements student and teacher are elements whose types are NameType. A change in the type Nametype affects both the elements student and teacher. This kind of dependency also causes coupling [7].

IV. Inheritance

In Object Oriented context the *Inheritance coupling* means that the coupling of two classes when one class is a subclass of another [8]. The coupling of this type is through data members that are inherited from a parent class. In Xml schema the coupling due to inheritance occurs when a data type is derived from another data type. The type derivation allows creating the base type from which other types can be derived. The types can be derived by means of Type extension, Type restriction, Groups and Redefine [6].

A. *Type Extension*

In Type extension pattern, a complex type can be extended to add some number of attribute or element declarations. In the following example (Example-4) the Extended Student type derives the student type which act as a base type. The Extended student contains the attributes grade, level and birth-date along with the elements Name and Major which are derived from the Student type. There is a coupling between student and Extended Student.

```
<xs:complexType name="Student">
   <xs:sequence>
   <xs:element Name="name" type=”xs:string”/>
   <xs:element Major= “major" type = “xs:string”/>
    </xs:sequence>
 </xs:complexType>
 <xs:complexType name="ExtendedStudent">
   <xs:complexContent>
      <xs:extension base="my:Student">
        <xs:attribute name="grade" type="xs:string"/>
       <xs:attribute name="level" type="xs:string"/>
       <xs:attribute name="birth-date" type="xs:date"/>
     </xs:extension>
   </xs:complexContent>
</xs:complexType>
```

Complex Type Extension (Example – 4)

B. *Type Restriction*

The restriction type pattern also used to derive a new type from the base type. But the occurrence of base elements can be limited or restricted. Both simple and complex type can be restricted. The following is the example (Example – 5) for deriving data types by means of type restriction from the base type Item.

```
<xs:complexType  name="Item">
  <xs:sequence>
      <xs:element name="number" type="xs:string"/>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="size" type="my:Size" minOccurs="0"/>
      <xs:element name="color" type="my:Colors" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
We can restrict the Item as:
<xs:complexType  name="LimitItem">
   <xs:complexContent>
     <xs:restriction base="my:Item">
     <xs:sequence>
       <xs:element name="number" type="xs:string"/>
       <xs:element name="name" type="xs:string"/>
    </xs:sequence>
   </xs:restriction>
  </xs:complexContent>
</xs:complexType>
```

Complex Type Restriction (Example – 5)

A simple type can also be used to create a new type through restriction. The following is the example for simple type restriction.

```
    <xs:element name="bike" type="BikeType"/>
<xs:simpleType name="BikeType">
 <xs:restriction  base="xs:string">
   <xs:enumeration value="Honda"/>
   <xs:enumeration value="Activa"/>
   <xs:enumeration  value="Kinetic"/>
 </xs:restriction>
</xs:simpleType>
```

Simple Type Restriction (Example – 6)

C. Groups

The collections of elements are allowed to be referenced and inherited using Groups in XML schema. Once a list of elements have defined as a group, it can be referred anywhere within the xml schema. In the given example (Example-7), once address group is referenced, address element gets four child elements namely StreetAddress, City, state and PostalCode. In the same fashion a group of attributes can also be inherited. Here we can say that element Address depends on the AddressGroup.

```
<xs: element name="Address">
    <xs:complexType>
    <xs:sequence>
         <xs:group ref="AddressGroup"/>
    </xs:sequence>
   </xs:complexType>
</xs:element>
<xs:group name="AddressGroup">
<xs:sequence>
          <xs:element name="StreetAddress" type="xs:string"/>
          <xs:element name="City" type="xs:string"/>
          <xs:element name="State" type="xs:string"/>
          <xs:element name="PostalCode" type="xs:string"/>
</xs:sequence>
</xs:group>
```

(Example-7)

In the above mentioned ways, the types are inherited. The advantage of inheritance is attaining of reuse. But at the same time, due to the derivation of one type (sub Type) from the other type (base type), the inheritance increases coupling between the type definitions. For a good design of service interface it should be loosely coupled. In order to reduce the coupling effect, the usage of inheritance derivation should be minimized.

V. MODULES

The important schema declarations are import, include and redefine which supports modular service interface[9]. These definitions bring definitions and declarations of external schema into the current schema. The “include” brings all definitions and declarations of an external schema whose target namespace is the namespace of current schema. It is usually used to extend the existing schema. The “redefine” does the same as “include” except it is used to redefine the available data type definitions of included schema document. The “import” also does the same as “include” except that external target namespace is different from the existing schema document. It brings all definitions and declarations of different name space schema documents to construct a new schema. In all these three methods, one xml schema documents depends on the structure of other schema documents and we can say that there is coupling between the documents.

VI. COUPLING AND REUSABILITY

During the designing of schemas the developers have to maintain a balance between reusability and coupling of schemas. If the intention is to design reusable schema, elements and types should be declared as global one. This will maximize reusability of schemas. The namespaces are exposed for reusability. But this causes the increment of coupling. (Figure – 1)

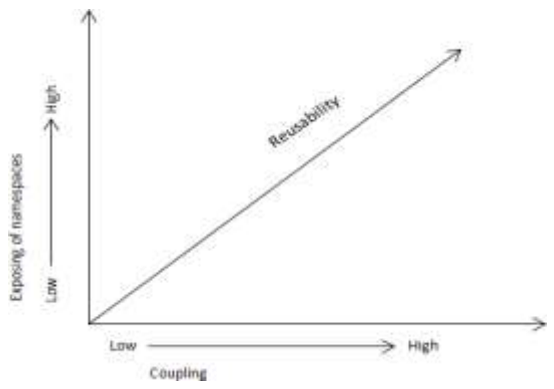

Figure – 1. Coupling Vs. Reusability

In the highly coupled schemas, the elements and types are highly interdependent. In this case it is difficult to do modifications and additions in future. Coupling is low, the reusability is also low. Coupling is high, the reusability is also high. Therefore the designing of reusable schema reduces future enhancement due to the presence of coupling. Another issue is performance of the service. More number of modules affects the performance of the system. There is a tradeoff between couplings, reusability and performance. Hence the selection of appropriate schema design pattern is important. Depending on the scope of the schema, the design pattern has to be selected.

VII. Conclusion

This paper identified the structural elements that causes coupling in xml service interface. We cannot say a particular design is the best one for service interface. However evaluating the service interface is very much important before going to develop the service. The service interface is the only known and measurable document to the consumer. Therefore service design characteristics in service interface should be evaluated. Our future work is proposing metrics for evaluation of service interface. The identified factors in this paper will be taken to continue our work.

References

[1] Michael Rosen, Boris Lublinsky, Kevi T. Smith and Marc J. Balcer, “Applied SOA”, Wiley India Edition, 2008
[2] Antony Reynolds, Matt Wright, “ORACLE SOA suite 11g R1 Developer’s Guide “, Pack Publishing Ltd., 2010
[3] http://www.ibm.com/developerworks/library/x-schemascope/.
[4] Dilek Basci and Sanjay Misra, “Measuring and Evaluating a Design Complexity Metric for XML Schema documents”, Journal of Information Science and Engineering, 25, 1405-1425 (2009).
[5] Thomas Erl, “SOA principles of Service Design”, Prentice Hall, 2009.
[6] Thomas Erl, Anish Karmarkar, Priscilla Walmsley, Hugo Haas, Umit Yalcinalp, Canyang Kevin Liu, David Orchard, Andre Tost, James Pasley “Web Service contract Design and Versioning for SOA” Prentice Hall, 2009.
[7] Eric Van der Vlist, “XML Schema”, Oreilly Publisher, 2002.
[8] Lionel C.Briand, Jhon W. Daly and Jurgen K. Wust, “ A Unified framework for coupling Measurement in Object Oriented Systems, IEEE transactions on Software Engineering, Vol -25, January – 1999.
[9] James Bean, “SOA and Web Services Interface Design, Principles, Techniques and Standards”, Morgan Kaufmann Publishers is an imprint of Elsevier,2010.