

WSDL-TC: Temporal Customization of Web Services

Preeti Marwaha¹, Hema Banati² and Punam Bedi³

¹ Department of Computer Science, AND College, University of Delhi,
Kalkaji, Delhi 110019, India
preeti_andc@yahoo.com

² Department of Computer Science, Dyal Singh College, University of Delhi,
Lodi Road, Delhi 110003, India
banatihema@hotmail.com

³ Department of Computer Science, University of Delhi,
Delhi 110007, India
pbedi@cs.du.ac.in

Abstract: Nowadays, companies recognize the need to be customer driven by providing superior service to satisfy customers' needs. But as customers and their needs grow increasing diverse, unnecessary cost and complexity are inevitably added to operations. Service providers discovered the new frontier in business competition: "Collaborative Customization." This approach follows three steps: first to conduct a dialogue with individual customers to help them articulate their needs; second, to identify the precise offering that fulfills those needs; and third, to make customized products for them. Web services deployed over the Web are accessible to a wider user base. Web services are designed and contracted to meet the need of large number of users. Many a times, multiple customizations of the base functionality is required to cater the need of multiple set of users. This forces service provider to deploy multiple Web Services customized for each set of users, which results in increasing cost of infrastructure and maintenance. Since, multiple versions of customized Web Services are deployed multiple times at different URLs, it is difficult and costlier to maintain, update and backup these services and their data. The objective of this work is to reduce the efforts and cost that resulted due to these multiple versions of the Web Services. We have extended WSDL and WSDL-T to WSDL-TC that aims at reducing the cost by maintaining the different collaborative customized versions of operations of the Web Service in a single deployment. The approach also manages access control of these operations to their respective groups. WSDL-TC being an extension of WSDL-T is capable of managing versions of each customized operation that resulted due to the changes in their business requirements over a period of time. WSDL-TC also eases the task of web service administrators as they have to manage the single instance instead of multiple instances of a Web service. Moreover, WSDL-TC based services when deployed in the cloud environment may help in achieving greater degree of multi-tenancy further reducing the cost for service producers.

Keywords: Web Services, WSDL, WSDL-Temporal Customization, WSDL-TC, Web Service Versioning.

I. Introduction

Web Services have become the proven technology enabling

the implementation of the Service Oriented Computing (SOC) paradigm. Web Services can be used to develop Web processes accessible within and across organizational boundaries. Web Services Description Language Version 2.0 [3] [4] (WSDL 2.0) is a W3C specifications that provides a model and an XML format for describing Web services. Although, the WSDL standard has become quite matured, there are still grey areas in Web services description that need to be addressed.

One such issue is management and deployment of various versions [8][16][17] of a Web service that are released over a period of time as an evolution or maturity process. These versions are released due to the ever changing demands of the organizations or due to the users' feedback. Banati et al. [1] described that the current specifications for Web service (WSDL 2.0) provide little coverage on managing these versions of Web services and extended WSDL to WSDL-Temporal (WSDL-T). WSDL-T enabled a single Web service running at a given URI to absorb the changes that occur from time to time. It also enabled the organization to publish the change without waiting for them to accumulate. In WSDL-T, the concept of linear temporal logic [18][19] as well as frame and slot versioning [5][20] is used for managing changes across multiple versions of Web service. WSDL-T allowed access to multiple versions of operations within Web service from the single URI. WSDL-T introduced two new attributes for every relevant artifact such as element, operation, interface, endpoint, service of WSDL viz. validity, timeStamp. If a change occurs in any element of the Web service resulting in a new version of an element, a new element is given the same name appended with the next version number and appropriate values of validity and timeStamp attributes are set. Apart from these new attributes the scheme for naming of element is modified and version number is appended in the name with the delimiter # e.g. name#x.y.z where x.y.z denotes the version number according to the user defined version scheme. Generally, x in version number can be associated with the versioning at Description element level, y can be associated as versioning at operation/interface level and z can be associated at the last level i.e. change in the parameter

element of an operation. The validity attribute can have any one value from the validity set {latest, past, deleted, alwaysTrue} and timeStamp is set to current date time in the format defined in XML schema [6] [7] data type xs:datetime. If validity status of any artifact is latest, then it denotes that the respective artifact is a part of latest version. Validity status past denotes that recent version of that artifact is already defined and the validity of that recent artifact is set to latest. Except recent version, all the old versions will have validity set to past or deleted. Validity status deleted denotes that, the particular artifact has been deleted in a new version. Validity status alwaysTrue denotes that, the artifact will be present in all the versions of service. The elements with this validity status form the basic functionality of the service and contracted as not changeable.

Further, WSDL specifications do not talk much about customization of Web Service. In the software industry many software products exist that caters the broad need of various customers but require customization to meet their complete requirements. Most of these products like ERP solutions, mailing solutions, Management Information System (MIS) exist as web based solutions. These systems need to be customized and deployed for a particular client/Entity like companies, organizations etc.

Our approach WSDL-TC [2] enhances the capability of Web services so that multiple customized Web services along with their versions that may come over a period of time can be deployed from a single instance. Moreover in customization, only a part of the service needs to be modified according to the client(s) whereas majority of the application (base functionality) remains unchanged. At present, in a data centre environment when deploying customized versions of the web service the whole application is replicated and deployed for each client i.e. both customized part of the application as well as part of application which is same for all clients are replicated and deployed number of times, resulting in redundancy of common code. This in turn results in increased cost both in terms of infrastructure and maintenance. Thus it would be better to maintain a single copy of that part of the application which is common for all the clients i.e. base functionality. Our approach allows us to maintain customized versions of temporally versioned web services from a single deployment.

The rest of this paper is organized as follows: section 2 discusses the related work with respect to the version strategies and customization. Section 3 discusses how WSDL-Temporal (WSDL-T) tackles issues related with change management. Section 4 describes WSDL-Temporal Customization (WSDL-TC), an extension over WSDL-T. Annotations for WSDL-T and WSDL-TC are presented in section 5. A case study of Front Line Demonstration of crop technology has been taken in Section 6. Various scenarios that may exist are presented in section 7, Results and Discussions. Section 8 provides future line of action for our study while Section 9 concludes the paper with merits of the approach.

II. Related Work

Web Services Description Language Version 2.0 (WSDL 2.0) [3], [4] is a W3C specification that provides a model and an XML [6], [7] format for describing Web services. Brown [8] discussed and categorized change types made in WSDL document as backwards-compatible and on-backwards-compatible changes. For backwards-

compatible changes, the WSDL document can simply be updated in the repository from which it is made available to requestors, and the existing Web service may be updated. If non-backwards-compatible changes need to be made to a WSDL document, then the changed web service is considered as completely new web service and deployed at new URL. We present an approach where all the versions of artifacts are maintained in a single WSDL document. Both backwards-compatible or non-backwards-compatible changes can happen in existing WSDL document and old as well as new users continue to use the same web service.

Kaminski et al. [9] addressed the problem of simultaneous deployment of multiple versions of a web service in the face of independently developed unsupervised clients. They proposed to use a form of a design technique called chain of adapters to make version-related reconfiguration tasks safe. Our approach maintains the versions or changes in different artifacts within same service and single web service is needed to be deployed for all the versions of the artifacts. We are using the temporal logic to access the different versions of artifacts.

Endrei et al. [10] recommend keeping no more than two concurrent versions of a service running, and envisage a transition time of three months. They also advocate toggling the versions between two service URLs. Together with their other statements, this recommendation implies that they are assuming a deployment environment much more controlled than the Internet at large. In comparison to this, our approach is more flexible and service producer has the capability to maintain multiple versions. Also, sometimes users prefer to use the older version over the new one, which is possible with our approach. If the user wants to switch to the newer version, he can do that also.

A couple of white papers [11], [8], describe the ways to design various versions of the web services and issues involved in it. The works cited above are tackling only backwards compatible changes and are managing to tackle incompatible changes by changing the target namespace value resulting in multiple versions of same web service.

Bechara [12] describes use of a mediation layer for decoupling the consumer from the provider by using Oracle Service Bus as a mediation layer. This layer provides the functionalities of the Layer of Indirection pattern and can be applied to a variety of versioning tasks such as accessing and deploying multiple versions of a service provider at the same time, routing requests to the appropriate service end point based on the content or the requester, adapting requests and responses to maintain backward compatibility, deprecating or retiring services in a graceful manner. This approach makes versioning tasks a bit easy for both producer and consumer but the approach requires an additional overhead of maintaining a mediation layer for version information in addition to multiple deployments of web service.

Henry Been [13] proposes a lightweight and visible approach to enrich WSDL files with versioning information by providing minimal number of tags in an existing WSDL. Their aim was to make WSDL able to give versioning information in the easiest manner. Their approach also encourages consumers to upgrade quickly to the new version. But our aim is to manage the changes made to web service in such a way that new as well as old consumers can continue with the web service without having any overhead. On specifying the deprecation date the consumer is encouraged to move to a newer version of that web service but

our approach allows the users to continue to use the older version or to upgrade to a newer version according to the consumers wish. Users can continue to use temporally customised version available for them.

Juric et al. [14] also addressed the problem of versioning of web services and this version information is reflected in UDDI. We are providing customization of temporally available versions and maintaining the information of versions in a single WSDL-TC file. Since WSDL-TC is containing the version information of an artifact we are not reflecting this information in UDDI. So far we have discussed the different approaches for web service versioning provided by different authors. We have found few publications that address Customization of web services at WSDL level. Jian Cao et al. [15] proposed a model to deal with the challenges of service customization. In the above approaches for web service versioning, we have found limited work that addresses the issue of customized versioning of artifacts according to the need of the users. In the presented work we enhance WSDL for Collaborative Customization as well as for change management. According to this approach user based upon their needs and demands, can use any version of artifact customized for them by the service producers. Customers can also request for an artifact that is customized for some other Entity. We also provide versioning of artifacts of web services that are customized for a set of users. Any of the above cited work is not taking into account versioning of artifacts and customization simultaneously in a way comparable to the approach proposed in this paper.

III. Temporal Enhancements to Web Services through WSDL-T

In the dynamic business world, requirements change quite often resulting in change in web service. There can be compatible and incompatible types of changes between two consecutive versions. Addition of a new operation/interface is a compatible change whereas modification or deletion of an operation/interface is incompatible change. There can be one or more number of such changes. When a compatible change occurs then the clients accessing the older versions are not affected and can continue to use older version till they want to switch to the newer one. On the other hand, if an incompatible change occurs then the clients accessing the older versions are affected. At present, whenever an organization makes major changes, it deploys the new version at some other URI and all the versions are required to be maintained simultaneously for providing service to old as well as new clients until all old clients are migrated to new version or the organization announces to end the support for the older versions. Organizations, to the larger extent, have automated the process of migrating clients from the older versions to the new version but still it is a cumbersome job at the server side. Also, some users tend to resist migration to new version unless they see a major benefit or they are forced to do so. In addition to this, it is reported [refer old version web site] that many times the users perceive that the new versions are not better than the previous one and they switch back to older versions. Yahoo messenger 7.0 and 7.5.0.647 have over a million downloads from the "oldversion.com" website after the newer versions have been launched. Similar statistics are available for some other popular software. It is also experienced that switching back to older version is not easy as most of the time it is not

assisted with the automated process. These processes are developed by the organizations for migrating to new versions and not for switching back. Moreover, if there is a customized application developed by the client, that connects to multiple web services provided by the multiple organizations then it becomes more difficult to upgrade them within the time frame dictated by these organization running the multiple versions of the web services. In view of the above mentioned points and the demand of specific version from the users forces the provider of the service to maintain a good number of versions at the server side. Since, business requirements are ever changing, so as the change required in the web service fulfilling those requirements. Another issue related to change management is the decision of time of release of a new version. Organizations are forced to wait for the release of new version of the web service until a sufficient number of small and big changes accumulate. It is a tradeoff between the minimum time between two releases of a web service and number of changes/business requirement. In the present body of research work, we propose two types of extension: WSDL-T and WSDL-TC to solve some of the problems caused by the frequent changes in the web services. WSDL-T enables a single web service running at a given URI to absorb the changes that occur from time to time. It also enables the organization to publish the change without waiting for them to accumulate.

In the presented work, the concept of linear temporal logic [19] as well as frame and slot versioning is used for managing changes across multiple versions of web service. WSDL-Temporal allows access to multiple versions of operations within web service from the single URI. WSDL-T introduces two new attributes for every relevant artifact such as element, operation, interface, endpoint, service of WSDL viz.

1. validity
2. timeStamp

Apart from these new attributes the scheme for naming of element is modified and version number is appended in the name with the delimiter # e.g. name#x.y.z where x.y.z denotes the version number according to the user defined version scheme. Generally, x in version number can be associated with the versioning at Description element level, y can be associated as versioning at operation/interface level and z can be associated at the last level i.e. change in the parameter element of an operation. The validity attribute can have any one value from the validity set {LATEST, PAST, DELETED, ALWAYSTRUE} and timeStamp is set to current date time in the format defined in XML schema data type xs:datetime.

LATEST - If validity status of any artifact is LATEST, then it denotes that the respective artifact is a part of LATEST version.

PAST - Validity status PAST denotes that recent version of that artifact is already defined and the validity of that recent artifact is set to LATEST. Except recent version, all the old versions will have validity set to PAST or DELETED.

DELETED - Validity status DELETED denotes that, the particular artifact has been DELETED in a new version.

ALWAYSTRUE - Validity Status ALWAYSTRUE denotes that, the artifact will be present in all the versions of service. The elements with this validity status form the

basic functionality of the service and contracted as not changeable.

If a change occurs in any element of the web service resulting in a new version of an element, a new element is given the same name appended with the next version number and appropriate values of validity and timeStamp attributes are set. If the change occurs at the element level like change in the number of arguments of an operation, then slot versioning is implied and only that particular element is added or deleted. If the change is at operation/interface or at service level like change in the name of the operation, then frame versioning is implied and the complete operation/interface or service is replicated with appropriate validity status along with time stamp. It is evident that choice of using slot or frame versioning is left to the designer of the web service according to the change occurred.

Thus we define WSDL-T an XML based web service description file that contain following literals

$$W_T = \{A, V, v, t\}$$

$A \in \{\text{Operation, Port Type, Message, Element}\}$
 $V = [\text{Name}] \# [0-9].[0-9].[0-9] \dots$
 $v \in \{\text{PAST, LATEST, DELETED, ALWAYS TRUE}\}$
 $t = \text{valid Date-Time}$

Where “A” is an artifact name, “V” denotes a version number associated with each artifact. Validity “v” and Time-stamp “t” is also included to each artifact. Validity and Time stamp will change as the changes occur in the artifact. In this section, we discuss different type of changes and how WSDL-Temporal tackles them.

Addition of a new operation: The most frequent change in web service is addition of new operations. This is a compatible change and does not affect the existing clients to access the new version of web service as all the older operations are still supported by the web service. In WSDL-Temporal, the new operation is given the same name as the current one with a version number appended to it and its validity status is set to LATEST and timeStamp set to current value of date time.

Deletion of an existing operation: In this case one or more no. of existing operations has been deleted from the current version to generate the new version. In our approach, the operation to be deleted will be replicated and the new element will have validity set to DELETED and timeStamp set to current date and time and version no. set to next version no. i.e. name#x.op+1.z. The validity status of existing operation will be changed to PAST from LATEST. Change the name of operation: This change is equivalent to the addition of a new element and deletion of existing element. The new element will be added with the validity set to LATEST, timeStamp as current date and time and version no. set to name#d.1.0. The existing element to be deleted will be tackled as explained above.

Complete change in Web service: Over a period of time the web service has substantial number of new elements, numbers of existing operations are DELETED, number of parameters of remaining existing functions are changed either in data types or in number but it performs same business functions as the previous one. This can be done by versioning Description element. The scheme for versioning at description level is at the top level i.e. first part after # sign name#x.0.0. For example: name#1.0.0., name#2.0.0. The description block will be replicated above the existing

description block and all the elements/operations/interfaces contained in it will have validity attribute set to LATEST or ALWAYS TRUE and timeStamp attribute set to current time stamp with version number set to name#x.0.0. WSDL-Temporal allow old clients to continue to use the older version of element/operation/interface within a web service without upgrading to the newer version of the element or operation whereas new clients have an option to choose among different versions according to their preferences. If old clients wish to upgrade to the newer version, they can do so. It is also evident that over a period of time no. of small changes will accumulate and there will be need to accumulate all small changes into next release of the Web service. Description versioning will be helpful in merging all these changes and combining them into the next version.

```

<!-- sample pseudo-schema of WSDL 2.0 -->
<defined_element
  required_attribute_of_type_string="xs:string"
  optional_attribute_of_type_int="xs:int"? >
  <required_element />
  <optional_element />?
  <one_or_more_of_these_elements />+
  [ <choice_1 /> | <choice_2 /> ]*
</defined_element>

```

(a)

```

<!-- sample pseudo-schema of WSDL-Temporal-->
<defined_element
  required_attribute_of_type_string="xs:string"
  required_attribute_of_type_string="xs:string"
  required_attribute_of_type_datetime="xs:datetime"
  optional_attribute_of_type_int="xs:int"? >
  [<required_element />]+
  <optional_element />?
  <one_or_more_of_these_elements />+
  [ <choice_1 /> | <choice_2 /> ]*
</defined_element>

```

(b)

Figure 1. Sample pseudo-schema of WSDL and WSDL-Temporal

WSDL 2.0 provides BNF Pseudo-Schemas for each of its component. They use BNF-style conventions for attributes and elements: “?” denotes optionality (i.e. zero or one occurrences), “*” denotes zero or more occurrences, “+” one or more occurrences, “[” and “]” are used to form groups, and “|” represents choice. Attributes are conventionally assigned a value which corresponds to their type, as defined in the normative schema. Elements with simple content are conventionally assigned a value which corresponds to the type of their content, as defined in the normative schema. Following is the BNF pseudo schema of WSDL2.0 and WSDL-Temporal. We have introduced two new attributes of type string for each valid element. The WSDL-Temporal follows the syntax defined by BNF pseudo-schema in Figure 1.

In Table 1, A_i denotes the artifact such as element/operation/interface, V^i denotes the version of element/operation/interface at timeStamp t_i . According to the change, the validity status of the A_i can be set to LATEST $_i$, PAST $_i$ or DELETED $_i$ at timeStamp t_i .

Table 1. Validity status of Various Versions of Artifact

	A ₀	A ₁	A ₂
V ¹	Latest _{t₁}		
V ²	Latest _{t₁}	Latest _{t₂} -> Past _{t₃}	
V ³	Latest _{t₁}	Deleted _{t₃}	Latest _{t₃}

Let's assume V¹ is the current version and A₀ is present in version V¹ with validity of A₀ set to LATEST at timeStamp t₁. At timeStamp t₂, there is a change of type addition and as a result A₁ is added at time stamp t₂ in version V². Thus the validity status of A₂ is set to LATEST at t₂ whereas A₁ is not changed. At timeStamp t₃, two changes occur in version V³. First change is addition of E₂ and second change is deletion of A₁. Thus validity status of A₁ is set to DELETED at timeStamp t₃ and validity status of E₁ is set to PAST at timeStamp t₂. A₂ is set to LATEST at timeStamp t₃ whereas A₀ is not changed in version V³.

WSDL-Temporal allow old clients to continue to use the older version of an artifact within a web service without upgrading to the newer version of the element or operation whereas new clients have an option to choose among different versions according to their preferences. If old clients wish to upgrade to the newer version, they can do so. It is also evident that over a period of time number of small changes will accumulate and there will be need to accumulate all small changes into next release of the Web service.

IV. Temporal Customization of Web Services through WSDL-TC

Traditionally, each customized version of a web service is treated as a different web service and is deployed separately. In this section, we extend WSDL-T to WSDL-TC, through which it is possible to customize temporal web service for multiple Entities running from a single Web service instance. Here, an Entity denotes the set of users with same requirements. An Entity may also be a set of users categorized on the basis of access rights/privileges assigned to them. WSDL-TC based web service allows all customized versions of operations to be run or deployed at single URL. By using this customization approach in WSDL-T it is possible to customize any valid version of the artifact, available at a particular time for any client. High degree of configurability, enable Entities to create their own workflow within the service. The approach also isolates and maintains security among various Entities so that an Entity cannot have access to the operations not authorized for them.

We have introduced a <customization> tag in the WSDL-TC, that contains one or more <Entity> tag(s), which specify the Entities/clients for whom temporal web services are customized. The name attribute in the <Entity> tag can be any qualified name and the value attribute in the <Entity> tag can be assigned with the user defined Entity name. The user defined Entity name assigned to the value attribute is used to create logical bundle of artifacts for a particular Entity. Figure 2 shows the customization tag appears in the beginning of the WSDL-TC file just after the Description and Documentation tags.

```
<wsdltc:customization>
<wsdltc:Entity name="anyQualifiedName" value=
"Entityid1" validity = "latest|past|alwaysTrue|Deleted"
timestamp="date-time" />
<wsdltc:Entity name="anyQualifiedName" value=
"Entityid2" validity = "latest|past|alwaysTrue|Deleted"
timestamp="date-time" />
..
.</wsdltc:customization>
```

Figure 2. WSDL-TC Customization Tag

The artifacts that need to be customized can have one or more <EntitySet> tags defined in their scope. EntitySet refers to the collection of Entities that share same customization. The <EntitySet> tag should have one <Entity> tag defined directly in its scope. This <Entity> defines a primary Entity. The <Entity> tag contains the required set of elements of WSDL that are usually defined in that artifact e.g. Input, Output, Outfault etc. in Operation artifact.

If a customized artifact is required by multiple entities then one can assign the same artifact to them without redefining it. This can be achieved by <AlsoApplicableTo> tag defined under <EntitySet> tag. The <AlsoApplicableTo> tag comprises of one or more <Entity> tag(s), which define the Entities for which the customized artifact is available, these <Entity> tag(s) defines secondary entities. Figure 3 shows the snippet of WSDL-TC where operation named "op1#1.0.0" is first defined for certain group of users, which we called the "base function". Then we have customized the same operation for Entity "Entity1" (primary Entity) which is made a part of EntitySet "S1". The same customization is needed by other Entities "Entity2" and "Entity3" then these two entities are made a part of same EntitySet. Here, we will define two <Entity> tags under <AlsoApplicableTo> tag for entities "Entity2" and "Entity3" respectively (secondary entities). Code under the scope of an artifact but not under the scope of any <EntitySet> tag is noncustomized version of the artifact and is called a "base function" which is available to all the entities except entities who have defined their own customization(s). It also means that the tags introduced by WSDL-TC are only applicable for customization and versioning and WSDL-TC converges with WSDL for normal usage. This customization can be achieved for different artifacts. Here we are limiting till operation level. Artifacts that are not customized for any Entity are accessible to all the entities. Each Entity in this way has a sum of common artifacts as well as customized artifacts to meet its requirements.

The artifacts that need to be customized can have one or more <EntitySet> tags defined in their scope. EntitySet refers to the collection of Entities that share same customization. The <EntitySet> tag should have one <Entity> tag defined directly in its scope. This <Entity> defines a primary Entity. The <Entity> tag contains the required set of elements of WSDL that are usually defined in that artifact e.g. Input, Output, Outfault etc. in Operation artifact. If a customized artifact is required by multiple Entities then one can assign the same artifact to them without redefining it. This can be achieved by <AlsoApplicableTo> tag defined under <EntitySet> tag. The <AlsoApplicableTo> tag comprises of one or more <Entity> tag(s), which define the

Entities for which the customized artifact is available, these <Entity> tag(s) defines secondary entities. Figure 3 shows the snippet of WSDL-TC where operation named “op1#1.0.0” is first defined for certain group of users, which we called the “base function”. Then we have customized the same operation for Entity “Entity1” (primary Entity) which is made a part of EntitySet “S1”.The same customization is needed by other Entities “Entity2” and “Entity3” then these two entities are made a part of same EntitySet. Here, we will define two <Entity> tags under <AlsoApplicableTo> tag for entities “Entity2” and “Entity3” respectively (secondary entities). Code under the scope of an artifact but not under the scope of any <EntitySet> tag is non-customized version of the artifact and is called a “base function” which is available to all the entities except entities who have defined their own customization(s). It also means that the tags introduced by WSDL-TC are only applicable for customization and versioning and WSDL-TC converges with WSDL for normal usage. This customization can be achieved for different artifacts. Here, we are limiting till operation level. Artifacts

that are not customized for any entity are accessible to all the entities. Each entity in this way has a sum of common artifacts as well as customized artifacts to meet its requirements.

Definition: A WSDL-TC is defined as an XML file that contain following literals to describe temporally customized web services.

- $W_{TC} = \{A, V, v, t, S\}$
- $A \in \{\text{Port Type, Operation, Message, Element}\}$
- $V = [\text{Name}] \# [0-9].[0-9].[0-9]...$
- $v \in \{\text{PAST, LATEST, DELETED, ALWAYSTRUE}\}$
- $t = \text{valid Date-Time}$
- $S = \{\text{PE}\} \cup \{\text{SE}\}$

Where “A” is an artifact name, “V” is a version number which is associated with each artifact. Validity “v” and Time-stamp “t” is also appended to each artifact. “S” denotes the EntitySet which is union of Primary Entity (PE) and secondary Entities (SE).


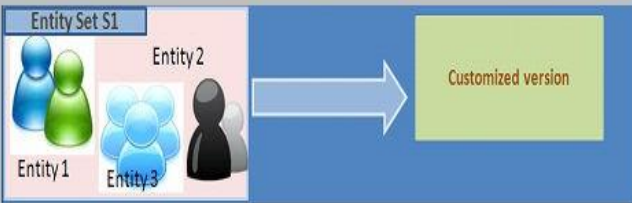
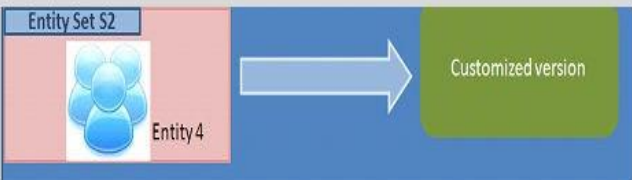
	<pre><wsdltc:operation name="op1#1.0.0" validity="latest" timestamp="07/11/2011 16:53:34" > <!--Base functionality goes here-- ></pre>
	<pre><wsdltc:EntitySet name="S1" validity="latest" timestamp="07/11/2011 16:53:34"> <wsdltc:Entity name="Entity1" value="Entity1" validity="latest" timestamp="07/11/2011 16:53:34"> <!-- Customized functionality goes here-- > </wsdltc:Entity> <wsdltc:alsoApplicableTo name="EntitySetApp1#1.0.0" validity="latest" timestamp="07/11/2011 16:53:34"> <wsdltc:Entity name="Entity2" value="Entity2" validity="latest" timestamp="07/11/2011 16:53:34"/> <wsdltc:Entity name="Entity3" value="Entity3" validity="latest" timestamp="07/11/2011 16:53:34"/> </wsdltc:alsoApplicableOn> </wsdltc:EntitySet></pre>
	<pre><wsdltc:EntitySet name="S2" validity="latest" timestamp="07/11/2011 16:53:34"> <wsdltc:Entity name="entity4" value="Entityid4" validity="latest" timestamp="07/11/2011 16:53:34"> <!--Customized functionality goes here--> </wsdltc:Entity> </wsdltc:EntitySet> </wsdl:operation></pre>

Figure. 3. Customization of Operation in WSDL-TC

```

@Retention(RetentionPolicy.RUNTIME)
@Target({TYPE})
public @interface WebServiceTC{
    public enum Validity {
        LATEST,
        PAST,
        DELETED,
        ALWAYSTRUE };
    String name() default "";
    String targetNamespace() default "";
    String serviceName() default "";
    String wsdlLocation() default "";
    String endpointInterface() default "";
    String portName() default "";
    Date timeStamp() default "";
    Validity validity() default
    Validity.LATEST;
};

@Retention(RetentionPolicy.RUNTIME)
@Target({TYPE,METHOD})
public @interface WebMethodTC{
    public enum Validity {
        LATEST,
        PAST,
        DELETED,
        ALWAYSTRUE };
    String OperationName() default "";
    String action() default "";
    String exclude() default "";
    Date timeStamp() default "";
    Validity validity() default
    Validity.LATEST;
};

@Retention(RetentionPolicy.RUNTIME)
@Target({TYPE})
public @interface EntitySet {
    public enum Validity {
        LATEST,
        PAST,
        DELETED,
        ALWAYSTRUE };
    String name() default "";
    Validity validity() default Validity.LATEST;
    Date timeStamp() default "";
    public Entity entity(); };

@Retention(RetentionPolicy.RUNTIME)
@Target({TYPE})
public @interface Entity {
    public enum Validity {
        LATEST,
        PAST,
        DELETED,
        ALWAYSTRUE };
    String name() default "";
    String value() default "";
    Validity validity() default Validity.LATEST;
    Date timeStamp() default "";
};

@Retention(RetentionPolicy.RUNTIME)
@Target({TYPE})
public @interface AlsoApplicableTo{
    public enum Validity {
        LATEST,
        PAST,
        DELETED,
        ALWAYSTRUE };
    String name() default "";
    Validity validity() default Validity.LATEST;
    Date timeStamp() default "";
    public Entity[] entities(); };

@Retention(RetentionPolicy.RUNTIME)
@Target({PARAMETER})
public @interface WebParamTC {
    public enum Mode {
        IN,
        OUT,
        INOUT
    };
    public enum Validity {
        LATEST,
        PAST,
        DELETED,
        ALWAYSTRUE };
    String name() default "";
    String targetNamespace() default "";
    Mode mode() default Mode.IN;
    boolean header() default false;
    String partName() default "";
    Validity validity() default
    Validity.LATEST;
    Date timeStamp() default "";
};

@Retention(value=RetentionPolicy.RUN
TIME)
@Target({METHOD})
public @interface WebResultTC {
    public enum Validity {
        LATEST,
        PAST,
        DELETED,
        ALWAYSTRUE };
    String name() default "return";
    String targetNamespace() default "";
    boolean header() default false;
    String partName() default "";
    Validity validity() default
    Validity.LATEST;
    Date timeStamp() default "";};

```

Figure 4. Proposed Java Annotation

V. Annotations of WSDL-T and WSDL-TC

To implement proposed extensions to WSDL, we have proposed modifications in annotations specified in JSR181 for `WebService`, `WebMethod`, `WebParam` and `WebResults` to associate validity and `timeStamp` to the artifacts of the web service (Figure 4). Two new members `timeStamp()` and `validity()` are added. Validity can have one of the following values: `Validity.LATEST`, `Validity.PAST`, `Validity.DELETED`, `Validity.ALWAYSTRUE`. The default assigned value is `Validity.LATEST`. The `timeStamp` is assigned a valid date- time value. We have defined new annotations `WebServiceTC`, `WebMethodTC`, `WebResultTC` and `WebParamTC` for change management and we have defined annotations `EntitySet`, `Entity` and `AlsoApplicableTo` to support the customization as defined in WSDL-TC. In

`EntitySet`, we have declared a nested annotation of type `Entity` to define a primary `Entity` for whom the customization is required. In `AlsoApplicableTo` annotation, we have declared an array of annotation `Entity` for all the secondary entities who share the same customization as that of primary `Entity`. We have developed a WSDL-TC writer, which can read annotations from java based web services and generate corresponding WSDL-TC file. Figure 5 shows the structure of java web services which uses annotations to define temporally customized artifacts of a service. The meta data in the annotations give the information regarding the version, validity, `timeStamp`, `EntitySet`, `Entities` (Primary as well as secondary) related to the artifact of a web service. Figure 6 shows the corresponding WSDL-TC schema generated for the above mentioned java web service format supporting temporal customization.

```

.....
@WebServiceTC(name="QName", targetNamespace="QName", validity="Validity.LATEST", timeStamp="date-time")

public class ClassName{

    @WebMethodTC(operationName="QName" validity=
    Validity.LATEST|Validity.PAST|Validity.DELETED|
    Validity.ALWAYSTRUE timeStamp="date-time")

    @EntitySet(name="QName", validity= Validity.LATEST |
    Validity.PAST | Validity.DELETED | Validity.ALWAYSTRUE, timeStamp="07/11/2011 16:53:34", @Entity(name="QName", value="", validity=
    Validity.LATEST | Validity.PAST | Validity.DELETED | Validity.ALWAYSTRUE, timeStamp="date-time"))

    @webResultTC(name="QName", targetNamespace="", partName="", Validity= Validity.LATEST | Validity.PAST | Validity.DELETED |
    Validity.ALWAYSTRUE timeStamp="date-time")
    @AlsoApplicableTo(name="QName", validity= Validity.LATEST | Validity.PAST | Validity.DELETED | Validity.ALWAYSTRUE,
    timeStamp="date-time", entities={
    @Entity(name="QName", value="", validity= Validity.LATEST | Validity.PAST | Validity.DELETED | Validity.ALWAYSTRUE, timeStamp="date-
    time"),
    @Entity(name="QName", value="", validity= Validity.LATEST | Validity.PAST | Validity.DELETED | Validity.ALWAYSTRUE, timeStamp="date-
    time"),
    ..... })

    public String method name (@WebParamTC(name="QName" validity= Validity.LATEST | Validity.PAST | Validity.DELETED |
    Validity.ALWAYSTRUE timeStamp="date-time")
    {
    //Customized operation definition
    }}

```

Figure. 5. Java Web Services Format supporting Temporal Customization

```

<description targetNamespace="xs:anyURI" validity="LATEST|PAST|DELETED|ALWAYSTRUE" timeStamp="xsd:DateTime" >
...
<interface name="xs:NCName" extends="list of xs:QName"? styleDefault="list of xs:anyURI"? validity="xs:NCName"
timeStamp="xsd:DateTime">

<fault name="xs:NCName" element="xs:QName"? validity="LATEST|PAST|DELETED|ALWAYSTRUE" timeStamp="xsd:DateTime">
</fault>*

<operation name="xs:NCName" pattern="xs:anyURI" style="list of xs:anyURI"? wsdl:safe="xs:Boolean"?
validity="LATEST|PAST|DELETED|ALWAYSTRUE" timeStamp="xsd:DateTime">

<wsdltc:EntitySet name="EntitySetid1" validity="LATEST|PAST|DELETED|ALWAYSTRUE" timeStamp="xsd:DateTime">
<wsdltc:Entity name="QName" value="Entity:d" validity="LATEST|PAST|DELETED|ALWAYSTRUE" timeStamp="xsd:DateTime">
<!-- Customized functionality goes here -->
</wsdltc:Entity>
<wsdltc:AlsoApplicableTo name="EntitySetApp1#1.0.0" validity="LATEST|PAST|DELETED|ALWAYSTRUE" timeStamp="xsd:DateTime">
<wsdltc:Entity name="QName" value="Entity2" validity="LATEST|PAST|DELETED|ALWAYSTRUE" timeStamp="xsd:DateTime"/>
<wsdltc:Entity name="QName" value="Entity3" validity="LATEST|PAST|DELETED|ALWAYSTRUE" timeStamp="xsd:DateTime"/>
</wsdltc:AlsoApplicableTo>
</wsdltc:EntitySet>
<input messageLabel="xs:NCName"? element="union of xs:QName, xs:Token"? validity="LATEST|PAST|DELETED|ALWAYSTRUE"
timeStamp="Date Time"> </input>*

<output messageLabel="xs:NCName"? element="union of xs:QName, xs:Token"? validity="LATEST|PAST|DELETED|ALWAYSTRUE"
timeStamp="Date Time"> </output>*

<infault ref="xs:QName" messageLabel="xs:NCName"? validity="LATEST|PAST|DELETED|ALWAYSTRUE" timeStamp="Date Time">
</infault>*

<outfault ref="xs:QName" messageLabel="xs:NCName"? validity="LATEST|PAST|DELETED|ALWAYSTRUE" timeStamp="Date Time">
</outfault>*
</operation>*
</interface>*
...
</description>

```

Figure. 6. WSDL-TC Structure


```

<operation name="FLD#1.0.0" validity="PAST" timeStamp="11-01-2010 14:20:08">
.
.
<operation name="FLD#1.0.0" validity="PAST" timeStamp="11-01-2010 14:20:08">
<!--base functionality initially designed for MaizeFLD -->
<wsdltc:EntitySet name="FLD1" validity="LATEST" timeStamp="11-01-2010 14:20:08">
<wsdltc:Entity name="RiceFLD" value="RiceFLD" validity="LATEST" timeStamp="11-01-2010 14:20:08">
<!-- Customized functionality for Rice FLD goes here -->
</wsdltc:Entity>
<wsdltc:AlsoApplicableTo name="ATFLD" validity="LATEST" timeStamp="11-06-2010 17:40:10">
<!--same Rice FLD Customization for Wheat FLD -->
<wsdltc:Entity name="WheatFLD" value="WheatFLD" validity="LATEST" timeStamp="11-06-2010 17:40:10"/>
</wsdltc:AlsoApplicableTo>
</wsdltc:EntitySet>
</operation>
<operation name="FLD#1.0.1" validity="LATEST" timeStamp="12-10-2010 17:20:08">
<!--new version of operation FLD#1.0.0-->

</operation>

```

Figure 7. WSDL-TC snippet for FLD Services

```

package mypack;
import ...
@WebServiceTC( name = "FLDWS", targetNamespace="http://www.wsdl-tc.info/spec", validity=Validity.LATEST,
timeStamp="11-01-2010 14:20:08")

public class ClassName{

@WebmethodTC(operationName = "FLD#1.0.0" validity="PAST" timeStamp="11-01-2010 14:20:08")

@EntitySet(name="FLD1",validity=Validity.LATEST,timeStamp="11-01-2010 14:20:08", @Entity(name="RiceFLD ",
value="RiceFLD "validity="LATEST", timeStamp="11-01-2010 14:20:08"))

@AlsoApplicableTo(name="ATFLD ",validity=Validity.LATEST,timeStamp="11-06-2010 17:40:10", entities={
@Entity(name="WheatFLD", value="WheatFLD"validity="LATEST", timeStamp="11-06-2010 17:40:10")})

public String FLD100 (@WebParam(name=" ..."))
{
//Customized operation definition
}}

```

Figure 8. Java code snippet for FLD Web Service

VI. Case Study

We have implemented WSDL-TC web services for FLD (Frontline Demonstration) for different crops like Maize, Rice, Wheat etc. Frontline Demonstration is a participatory research, emphasizing scientist-farmer interaction, refining and validating research findings, developing leadership amongst farmers for multiplier effect to horizontally disseminate technology. The FLDs provide an effective learning situation as the farmers observe the technologies, practice it and interact with the scientists and extension functionaries. It is very necessary to record the observation and get the feedback from the farmers and the extension workers for all the FLD experiments. It also helps in analyzing the FLD experiment as well as FLD program as a whole. Since, the Internet connectivity is not readily available at farmers' field, so a Web service based approach is used for collecting the data from the Maize farmers' field. Also, FLD performas' tend to change a bit over time and for different

crops, so a new approach based on WSDL-Temporal Customization Web Service has been proposed in this work. The data and feedback collection system has been designed using WSDL-TC based Web service architecture.

Initially, we designed web service and their clients for FLD for Maize which we called as Base function. This function is non-customized function which is available to all the clients of this web service. Then we extended our work for FLD for Rice crop. There we need to incorporate some changes according to new crop. Thus, we modified some operations of the existing service and its clients resulting in a new customized version of an artifact within the same service. With time some functionality of FLD for Maize crop changes, which resulted in a new version of the operation in the service. Now, two versions of operation FLD for maize exist and the base function is also customized for FLD for Rice crop. FLD for wheat is exactly same as FLD for Rice thus same operation can be accessed for wheat FLD. Hence, Wheat FLD now becomes the secondary Entity and we put it within

alsoApplicableTo tag (shown in Figure 5). Figure 6 is the corresponding java code for WSDL-TC based web service.

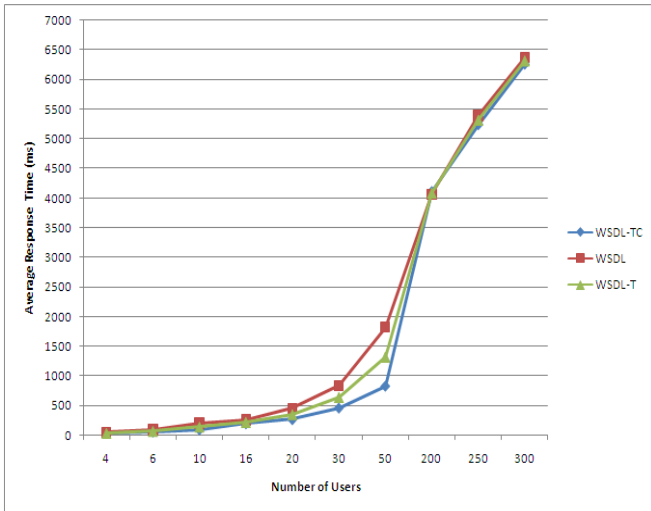


Figure 9. Average Response Time of WSDL, WSDL-T and WSDL-TC

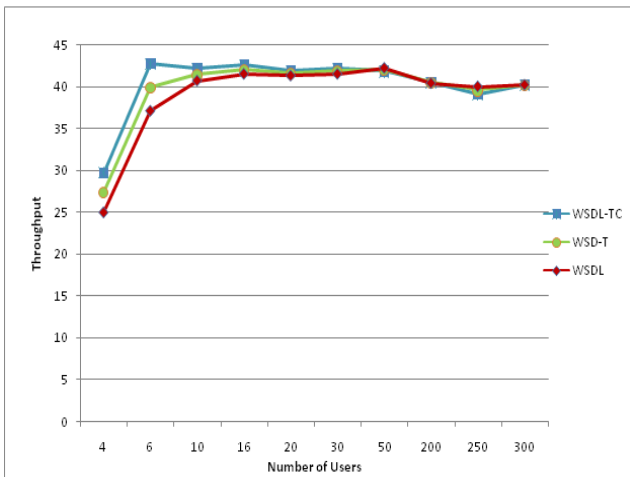


Figure 10. Throughput of WSDL, WSDL-T and WSDL-TC.

Using Apache Jmeter, we have run and compared the throughput and average response time (for different number of concurrent requests) of the standard web service with our temporal and temporally customized web services and reached to a conclusion that response time and the throughput of both WSDL-T and WSDL-TC are comparable to WSDL as shown in Figure 9. Figure 9 shows that in the initial phase as the number of concurrent users are increased the average

response time increased gradually. Later on as number of users crossed the maximum load capacity limit of the server, the average response time increased proportionately. The experiment shows that WSDL and WSDL-T/TC behaved in similar fashion, although the WSDL-T/TC showed a little improvement especially in the region of maximum load capacity. This is due to the fact that till the system resources are available, the user requests are served up to the satisfaction levels. After saturation level is reached and there are no more resources left, more number of users are put to wait state. So, the average response time rises sharply.

The second graph in Figure 10 shows that although the average response time increases sharply after a point, the throughput increases initially and remains almost constant afterwards. Thus, the graphs in Figure 9 and Figure 10 show that WSDL-T/TC do not degrade the average response time or throughput and the overhead required to process the multiple versions is minimal. It means that when WSDL-TC based web services are deployed, the service producers may deploy multiple versions for their multiple clients from a single instance. This in turn has a clear reduction in terms of infrastructure requirements as number of instances per service is reduced to one. Man power requirements for managing and taking back-up of multiple versions are also reduced because there is only single instance per service is required to be deployed. It also allows ease in patch management as the security patches or bug fixing in the non-customized and non-versioned segment of the web service is required to be done at a single place rather than in all the versions.

VII. Results and Discussion

WSDL-T and WSDL-TC allows us to maintain temporal and temporally customized version of the timeStamp of a web service. In this section, we show the various scenarios that are tackled over time in WSDL-T and WSDL-TC file. In Table 2, first column details the scenario at a particular time t_m and also discusses its solution. Second column shows the operations available in the service along with their validity and timeStamp Status. $O_{i\#v_j}(V, T)$ denotes j^{th} version of i^{th} operation with validity V and timeStamp T . $V=\{P|D|A|L\}$ and $T=\text{date and time value}$. The WSDL-TC allows customization of artifacts for multiple entities. It means that both customization and versions brought out by these entities are considered. Case Scenarios of WSDL-TC are discussed in Table 3.

Table 2. WSDL-T Case Scenarios

Solution Description	Operation#version_num(Validity,timeStamp)
Scenario: At time t_0 initial version of all operations $\{O_1, O_2, O_3\}$ exist.	
This is the initial version of web service with validity and timeStamp.	$\{(O_1\#v_1)L, t_0, (O_2\#v_1)L, t_0, (O_3\#v_1)L, t_0\}$
Scenario: At time t_1 new Operation O_n is added.	
Now first version of a new operation is added with validity LATEST and timeStamp t_1 .	$\{(O_1\#v_1)L, t_0, (O_2\#v_1)L, t_0, (O_3\#v_1)L, t_0, (O_n\#v_1)L, t_1\}$
Scenario: At time t_2 , version v_1 of operation O_1 is to be modified to a new version i.e. version v_2 of operation O_1 .	

Since operation O_1 is modified, the validity of version v_1 of operation O_1 is set to PAST and new version v_2 of operation O_1 is added with LATEST validity and timeStamp t_2 .	$\{(O_1\#v_1)P, t_0, (O_2\#v_1)L, t_0, (O_3\#v_1)L, t_0, (O_n\#v_1)L, t_1, (O_1\#v_2)L, t_2\}$
Scenario: At time t_3 a new operation with validity ALWAYSTRUE is added.	
Validity of operation O_4 is set to ALWAYSTRUE therefore this operation will always available and no further modification is allowed to it.	$\{(O_1\#v_1)P, t_0, (O_2\#v_1)L, t_0, (O_3\#v_1)L, t_0, (O_n\#v_1)L, t_1, (O_1\#v_2)L, t_2, (O_4\#v_1)A, t_3\}$
Scenario: At time t_4 , operation O_3 is deleted	
A same new version of the operation i.e. $O_3\#v_2$ is created with timeStamp t_4 and the validity status is set to DELETED and timeStamp set to t_4 whereas validity status of EntitySet $O_3\#v_1$ is changed to PAST.	$\{(O_1\#v_1)P, t_0, (O_2\#v_1)L, t_0, (O_3\#v_1)P, t_0, (O_n\#v_1)L, t_1, (O_1\#v_2)L, t_2, (O_4\#v_1)A, t_3, (O_3\#v_2)D, t_4\}$

Table 3. WSDL-TC Case Scenarios

Solution Description	Parts of Operation	EntitySet	Entity
Scenario: At time t_0 version v_1 of operation O_1 exists for all the users.			
This is the base functionality defined before customization and is as per the specifications of WSDL.	$O_1\#v_1(L, t_0)$		
Scenario: At time t_1 Entity E_1 wants to customize version v_1 of operation O_1 .			
Version v_1 of Entity i.e. $E_1\#v_1$ within version v_1 of EntitySet i.e. $S_1\#v_1$ is created within the scope of $O_1\#v_1$ with timeStamp t_1 . Now, we have base functionality defined within $O_1\#v_1$ followed by customized functionality for Entity E_1 contained in EntitySet S_1 . So, $O_1\#v_1$ is transformed to $(O_1\#v_1)S_1$.	$O_1\#v_1(L, t_0)$		
	$(O_1\#v_1)S_1(L, t_1)$	$S_1\#v_1(L, t_1)$	$E_1\#v_1(L, t_1)$
Scenario: At time t_2 version v_1 of operation O_1 which is already customized for Entity E_1 , is required by Entity E_2 without any change.			
Since, the customization done in $(O_1\#v_1)S_1$ also applies to E_2 , $E_2\#v_1$ is defined with timeStamp t_2 within same EntitySet S_1 under the AlsoApplicableTo tag. Thus same operation denoted by $(O_1\#v_1)S_1$ is available to both E_1 and E_2 .	$O_1\#v_1(L, t_0)$		
	$(O_1\#v_1)S_1(L, t_1)$	$S_1\#v_1(L, t_1)$	$E_1\#v_1(L, t_1)$
			$E_2\#v_1(L, t_2)$
Scenario: At time t_3 , another Entity E_3 wants its own customization for the version v_1 of operation O_1 .			
Version v_1 of Entity i.e. $E_3\#v_1$ within version v_1 of EntitySet i.e. $S_2\#v_1$ is created within the scope of $O_1\#v_1$ with timeStamp t_3 . So, $(O_1\#v_1)S_1$ is transformed to $(O_1\#v_1)S_2$.	$O_1\#v_1(L, t_0)$		
	$(O_1\#v_1)S_1(L, t_1)$	$S_1\#v_1(L, t_2)$	$E_1\#v_1(L, t_2)$
			$E_2\#v_1(L, t_2)$
	$(O_1\#v_1)S_2(L, t_3)$	$S_2\#v_1(L, t_3)$	$E_3\#v_1(L, t_3)$
Scenario: At time t_4 , version v_1 of operation O_1 is to be modified to a new version i.e. version v_2 of operation O_1 .			
Since, the base functionality has changed new version $O_1\#v_2$ of operation O_1 is created with timeStamp t_4 . E_1 ,	$O_1\#v_1(P, t_0)$		
E_2 and E_3 continued to use their same old customized operations. Validity status of $O_1\#v_1$ is	$(O_1\#v_1)S_1(L, t_1)$	$S_1\#v_1(L, t_2)$	$E_1\#v_1(L, t_2)$

now changed to PAST.			$E_2\#v_1(L, t_2)$
	$(O_1\#v_1)S_2(L, t_3)$	$S_2\#v_1(L, t_3)$	$E_3\#v_1(L, t_3)$
	$O_1\#v_2(L, t_4)$		
Scenario: At time t_5 , Entity E_2 wants its own customization of operation O_1 whereas Entity E_1 continues to use its customized version.			
A new EntitySet $S_3\#v_1$ with new version $E_2\#v_2$ of the Entity E_2 is created with timeStamp t_5 and validity status LATEST within the scope of LATEST version of O_1 i.e. $O_1\#v_2$. So, EntitySet $S_1\#v_1$ in $O_1\#v_1$ contains $E_1\#v_1$ along with Entity $E_2\#v_1$ with validity status changed to PAST.	$O_1\#v_1(P, t_0)$		
	$(O_1\#v_1)S_1(L, t_1)$	$S_1\#v_1(L, t_2)$	$E_1\#v_1(L, t_2)$
			$E_2\#v_1(P, t_2)$
	$(O_1\#v_1)S_2(L, t_3)$	$S_2\#v_1(L, t_3)$	$E_3\#v_1(L, t_3)$
	$O_1\#v_2(L, t_4)$		
	$(O_1\#v_2)S_3(L, t_5)$	$S_3\#v_1(L, t_5)$	$E_2\#v_2(L, t_5)$
Scenario: Entity E_3 wants new customization of operation O_1 altogether.			
A new version of the Entity $E_3\#v_2$ is created with timeStamp t_6 within the scope of EntitySet S_2 and the validity status set to LATEST whereas validity status of Entity $E_3\#v_1$ is changed to PAST.	$O_1\#v_1(P, t_0)$		
	$(O_1\#v_1)S_1(L, t_1)$	$S_1\#v_1(L, t_2)$	$E_1\#v_1(L, t_2)$
			$E_2\#v_1(P, t_2)$
	$(O_1\#v_1)S_2(L, t_3)$	$S_2\#v_1(L, t_3)$	$E_3\#v_1(P, t_3)$
	$O_1\#v_2(L, t_4)$		
	$(O_1\#v_2)S_3(L, t_5)$	$S_3\#v_1(L, t_5)$	$E_2\#v_2(L, t_5)$
	$(O_1\#v_1)S_2(L, t_3)$	$S_2\#v_1(L, t_3)$	$E_3\#v_2(L, t_6)$
Scenario: Entity E_3 does not want to continue the use of the operation O_1 or decided to use the latest base functionality.			
A same new version of the EntitySet S_2 i.e. $S_2\#v_2$ and Entity $E_3\#v_2$ is created with timeStamp t_6 and the validity status is set to DELETED whereas validity status of EntitySet $S_2\#v_1$ and Entity $E_3\#v_1$ is changed to PAST.	$O_1\#v_1(P, t_0)$		
	$(O_1\#v_1)S_1(L, t_1)$	$S_1\#v_1(L, t_2)$	$E_1\#v_1(L, t_2)$
			$E_2\#v_1(P, t_2)$
	$(O_1\#v_1)S_2(P, t_3)$	$S_2\#v_1(P, t_3)$	$E_3\#v_1(P, t_3)$
	$O_1\#v_2(L, t_4)$		
	$(O_1\#v_2)S_3(L, t_5)$	$S_3\#v_1(L, t_5)$	$E_2\#v_2(L, t_5)$
	$(O_1\#v_1)S_2(P, t_3)$	$S_2\#v_2(P, t_3)$	$E_3\#v_2(P, t_6)$
	$(O_1\#v_1)S_2(D, t_7)$	$S_2\#v_2(D, t_7)$	$E_3\#v_2(D, t_7)$

VIII. Future Work: WSDL-TC in Multi-Tenant Scenario

At present Data multi-tenancy is the most explored approach under multi-tenancy, and is often implemented on top of a database. Both Jacobs et al. [21] and Chong et al. [22] have outlined three main approaches for data management in a multitenant deployment: separate databases, shared database with separate schemas and shared database with shared schemas. Among these, the separate database approach gives each tenant its own database, the shared database with separate schema approach gives each tenant its own tables, and in the shared database with shared schema approach shares the same table among many tenants and enforces security at the next layer in the architecture. They make a continuum where more isolation means less sharing & less scalability and vice versa. Chong et al. [22] in one of the discussions of multitenant applications, have proposed a maturity model where higher numbers indicate higher level of resource sharing. For instance, level 1 provides an instance per tenant, level 2 provides a configurable instance per tenant, level 3 runs a single instance that serves all customers, and finally, level 4 enables level 3 to scale up by running multiple instances and load balancing to scale it up. Guo et al. [23] categorize multi-tenancy as single instance vs. multiple instance, where the former serves many users using the same instance whereas the latter serves users by running multiple instances using one of the aforementioned methods. Azeez et al. [24] designed and implemented multitenant SOA platform which allows users to run their current applications in a multi-tenant environment with minimal or no modifications. Configurability is provided through a management portal and via server-side tenant specific customizations. All the above cited work is based on WSDL and trying to achieve multi-tenancy by introducing new layers above WSDL. Our approach enhances WSDL to WSDL-TC for supporting multiple collaborative customization of a service that can be executed from a single instance which is in fact essence of multitenant concept. We believe that WSDL-TC will be helpful in achieving higher degree of multi-tenancy specially when taken together with the above said approaches such as defined by the Azeez et al. [24] The use of WSDL-TC in multitenant environment will also help in deploying multiple versions of the service customized for multiple set of users from a single instance. As part of the future work of this paper we will verify and study effects of WSDL-TC in multitenant cloud environment.

IX. Conclusion

Today, with every computer and potentially every application connected to Internet, there are increased requirements for interconnected and interdependent business-to-business applications that can facilitate business over the Internet. In a fast changing economy, the nature of business relationships constantly changes. Reorganization, mergers, and new business partnerships are relentlessly negotiated resulting in change in the business applications. The service oriented architecture ensures building dynamic, highly scalable and interoperable web applications. WSDL 2.0 is the present specifications for developing web services and has covered little on the change management and customization. In the presented work, WSDL has been extended to WSDL-TC. By using WSDL-TC, it is possible

to customize any valid version of the artifact, available at a particular time for any client. This enables to create customized functionality within a service for each Entity (set of users). One of the main advantages of this approach is operational benefit. Because all application code is in one place, it is much easier and cheaper to maintain, update and backup the service and its data. Also, in case of WSDL, if a patch is required to be applied for any bug fixing or security management then it needs to be applied at all the deployments making the process time consuming and extra efforts are required. Here, if the patch is in the operations that are not changed or customized then it needs to be applied at a single location irrespective of number of Entities that are using the service. Another advantage is the overall lower system resources required for running the single deployment as compared to multiple deployments. It is not necessary to have a dedicated version of the service resources and infrastructure for every Entity (set of users). This is a clear improvement in resources utilization, saves time and lowers overall cost. A case study on Front Line Demonstration has been implemented using WSDL-TC and it has been shown that the performance of WSDL and WSDL-TC based services are comparable with the additional benefits delivered by the WSDL-TC.

References

- [1] H. Banati, P. Bedi, P. Marwaha. "WSDL-Temporal: An approach for change management in Web Services". In *Proceedings of the IEEE International Conference on Uncertainty Reasoning and Knowledge Engineering*, pp. 44-49, 2012.
- [2] H. Banati, P. Bedi, P. Marwaha. "WSDL-TC: Collaborative customization of Web Services". In *Proceedings of the IEEE International Conference on Intelligent System Design and Applications (ISDA)*, pp. 692-697, 2012.
- [3] D. Booth, C. K. Liu. "Web Services Description Language (WSDL) Version 2.0 Part 0: Primer". *W3C Recommendation*, Available from: <http://www.w3.org/TR/2007/REC-wsdl20-primer-20070626>, 2007.
- [4] R. Chinnici, J. J. Moreau, A. Ryman, S. Weerawarana. "Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language". *W3C Recommendation*, Available from: <http://www.w3.org/TR/2007/REC-wsdl20-20070626>, 2007.
- [5] P. Bedi, K.D. Sharma, S. Kaushik. "Time dimension to frame systems". *Journal of Information Science and Technology*, II (3), pp. 212-228, 1993.
- [6] P. V. Biron, A. Malhotra. "XML Schema Part 2: Datatype". *W3C Recommendation*, Available from: <http://www.w3.org/TR/2002/WD-xmlschema-2-20010502/>, 2007.
- [7] T. Bray, J. Paoli, C.M. Sperberg-McQueen, E. Maler, F. Yergeau. "Extensible Markup Language (XML) 1.0 (Fourth Edition)". *W3C Recommendation*, Available from: <http://www.w3.org/TR/2006/REC-xml-20060816/>, 2006.
- [8] K. Brown, M. Ellis. "Best practices for Web services versioning". Available from: <http://www.ibm.com/developerworks/webservices/library/ws-version>, 2004.

- [9] P. Kaminski, M. Litoiu H. Muller. "A design technique for evolving web services". In *Proceedings of the Conference of the Center for Advanced Studies on Collaborative Research*, pp. 211-222, 2006.
- [10] M. Endrei, M. Gaon, J. Graham, K. Hogg, N. Mulholland. "Moving forward with Web services backwards compatibility". *Developer-Works*, Available from: <http://www-128.ibm.com/developerworks/java/library/ws-soa-backcomp/index.html?ca=drs-#resources>, 2006.
- [11] D. Parachuri, S. Mallick. "Service Versioning in SOA". Available from: <http://www.infosys.com/consulting/soa-services/white-papers/Documents/service-versioning-SOA1.pdf>, 2007.
- [12] G. Bechara. "Web Services Versioning". *Oracle Technology Network*, Available from: <http://www.oracle.com/technetwork/articles/web-services-versioning-094384.html>, 2008.
- [13] H. Been. "Extending WSDL with versioning information". In *Proceedings of the 14th Twente Student Conference on IT*, Available from: <http://referaat.cs.utwente.nl/conference/14/paper/7221/extending-wsdl-with-versioning-information.pdf>, 2011.
- [14] M. Juric, A. Sasa, B. Brumen I. Rozman. "WSDL and UDDI extensions for version support in web services". *Journal of Systems and Software*, LXXXII(8), pp.1326-1343, 2009,.
- [15] J. Cao, J. Wang, K. Lawb, S. Zhang, M. Li. "An Interactive service customization model". *Journal of Information and Software Technology*, XLVIII(4), pp. 280-296, 2009.
- [16] T. Erl, A. Karmarkar, P. Walmsley, H. Haas, U. Yalcinalp, C.K. Liu, D. Orchard, A. Tost, J. Pasley. *Web Service Contract Design & Versioning for SOA*, Prentice Hall, 2008.
- [17] J. Evdemon. "Principles of Service Design: Service Versioning". <http://msdn.microsoft.com/en-us/library/ms954726.aspx>, 2005.
- [18] J.F. Allen, "Maintaining Knowledge about Temporal Intervals". *Communications of the ACM*, XXVI, pp.832-843, 1983.
- [19] J.F. Allen, "Actions and Events in Interval Temporal Logic". *Journal of Logic and Computation*, 4, pp. 531-579, 1994.
- [20] P. Bedi. "A Unified Approach for Designing Frame-Based Systems". *Ph.D. Dissertation*, Dept. of Computer Science, University of Delhi, 1999.
- [21] D. Jacobs, S. Aulbach. "Ruminations on multi-tenant databases". In *Proceedings of BTW*, 2007.
- [22] F. Chong, G. Carraro, R. Wolter. "Multi-Tenant Data Architecture". *MSDN Library*, Microsoft Corporation, 2006.
- [23] C. J. Guo, W. Sun, Y. Huang, Z. H. Wang, B. Gao. "A Framework for Native Multi-Tenancy Application Development and Management". In *Proceedings of the 9th IEEE International Conference on E-Commerce Technology and The 4th IEEE International Conference on Enterprise Computing, E-Commerce and E-Services (CEC-EEE 2007)*, pp. 551-558, 2007.
- [24] A. Azeez, S. Perera, D. Gamage, R. Linton, P. Siriwardana, D. Leelaratne, S. Weerawarana, P. Fremantle. "Multi-Tenant SOA Middleware for Cloud Computing". In *Proceedings of IEEE 3rd International*

Conference on Cloud Computing, pp. 458-465, 2010.

Author Biographies



Preeti Marwaha is a Ph.D. scholar in the Department of Computer Science, University of Delhi. She is an Assistant Professor in the Department of Computer Science, A.N.D. College, University of Delhi. Her research interest includes Web Services and Composite Web Services, Semantic Web Services etc.



Dr. Hema Banati completed her Ph.D. (2006) after her Masters in Computer Applications(M.C.A) both from Department of Computer Science, University of Delhi, India. At present she is an Associate Professor in the Department of Computer Science, Dyal Singh College, University of Delhi. She has over 18 years of teaching experience to both undergraduate and postgraduate classes. Over the past decade she has been pursuing research in the areas of Web engineering, software engineering, Human Computer Interaction, Multi-agent systems, E-commerce and E-learning. She has many national and international publications to her credit.



Dr. Punam Bedi received her Ph.D. in Computer Science from the Department of Computer Science, University of Delhi, India in 1999 and her M.Tech. in Computer Science from IIT Delhi, India in 1986. She is an Associate Professor in the Department of Computer Science, University of Delhi. She has about 24 years of teaching and research experience and has published about 100 papers in National / International Journals / Conferences. Dr. Bedi is a member of AAAI, ACM, senior member of IEEE, and life member of Computer Society of India. Her research interests include Web Intelligence, Soft Computing, Semantic Web, Multi-agent Systems, Intelligent Information Systems, Intelligent Software Engineering, Intelligent User Interfaces, Requirement Engineering, Human Computer Interaction (HCI), Trust, Information Retrieval and Personalization.