A SERVICE VERSIONING MODEL FOR PERSONALIZED E-LEARNING SYSTEM

K. PALANIVEL
Computer Centre,
Pondicherry University, Puducherry - 605014, India
kpalani@yahoo.com
http://www.pondiuni.edu.in

S. KUPPUSWAMI
Department of Computer Science, School of Engineering,
Pondicherry University, Puducherry - 605014, India
skswami@yahoo.com
http://www.pondiuni.edu.in

Abstract:
Service Versioning is a hot topic that has generated a broad range of guidance from a variety of sources. Several Web services standardization efforts are underway but none of them address the problem of service versioning. Service versioning with metadata in e-learning system makes easy search, retrieval, import and evaluate. In E-learning systems, learning objects with metadata allow the learners to use quality educational contents filling their characteristic and teacher may use quality educational contents to structure their courses. A service versioning model in e-learning objects system with metadata satisfies the requirements for learning environment. In personalized e-learning systems, when more numbers of learner searching data the retrieval time is being delayed when the data volume is increasing continuously. This problem can be solved by designing and applying a proper service versioning model to personalized e-learning systems. Hence, it is proposed to design a service version model with metadata for personalized e-learning system. This proposed service versioning model satisfies some of the requirements like accessibility, interoperability, adaptability, durability and reusability in learning environment. Support multiple versions of a service isolate more expensive business behavior calls to specific versions so as to reduce impact to all of the other learners.

Keywords: E-learning, Learning Objects, Versioning Model, SOA, Web Service.

1. Introduction
Service-Orientation (SO) enables developing enterprise applications in a broader context application architectures are shifting away from monolithic solution silos into loosely-coupled infrastructures of autonomous services [1]. The service model is fractal, enabling application solutions to be constructed out of multiple services. This model enables the realization of loosely-coupled business processes, allowing enterprises to experiment with new processes or business models that are relatively free from the constraints of the underlying IT infrastructure.

Web Services-enabled systems can be linked together to create automated business processes-sharing data and functions. These can provide e-learning initiatives with reduced training administration overhead, integrated training data for Enterprise Resource Planning (ERP) reporting systems, simplified learner management, and cleaner user experiences [4]. Services should be deployed and versioned independently of the system in which they are deployed and consumed [14]. There are two forms of versioning: Forward compatibility means that newer versions of producers can be deployed without breaking existing consumers. Forward compatibility can be compensated for by adopting standards designed to ignore unrecognizable elements that may appear in the future. A service is backward compatible if it is compatible with earlier versions of itself—especially if these earlier versions are expected to be deprecated. The goal of backward compatibility is to ensure that consumers...
of older versions of a service can continue to interoperate with the new version. Backwards compatibility focuses on preserving existing service contracts while extending the service to add new features. Forward compatibility is much harder to achieve than backward compatibility since the service must be expected to gracefully interact with a number of unknown or unexpected features. The changes to services that may be required can be classified into four categories [2]. They are contract changes, address changes, binding changes, and implementation changes. These changes are called "breaking" and others are "nonbreaking."

1.1. Problem Definition

E-learning [3] involves the use of computers or electronic devices to transfer skills and knowledge. For achieving the adaptive learning, the material reducing the production costs to an acceptable amount. E-learning is often perceived as a group effort, where content authors, instructional designers, multimedia technicians, teachers, trainers, database administrator, and people from various other areas of expertise come together in order to serve a community of learners [1]. Authors or teacher/instructor create content, that learning content is main component to develop an e-learning system, reusability of learning object is adaptive learning material to serve learner.

The reuse of already existing learning material is fundamental. The ability to reuse existing learning objects enables the flexible recombination of them for new context’s or for learners with special needs. Applying versioning model for reuse of learning object in the learning system can easily handle multiple versions of objects. To support continuous reuse on the same learning object present versioning model that maintain all reuse learning objects as new version learning object and present metadata model for each new versioned learning objects to maintain all details of versioned information and stored in learning object repository (LOR).

A learning management system (LMS) [35] in under the control of an administration and it interacts with a run-time environment which is addressed by learners, who in turn may be coached by a trainer. For fast development, continuous research is required on controlling several aspects of the learning objects, such as the content design, back-end delivery and front-end presentation, in order to improve the quality of content delivery.

1.2. Solution to the Problem

Present metadata for the versioned learning object through the specification of the metadata and the Learning Object Metadata (LOM) provide a comprehensive suite of e-learning capabilities that enable interoperability, accessibility, and reusability of web-based learning content [2]. Both of these specifications contain support for capturing the fact that a relationship exists between two learning objects by using comparing technique that compare the two learning objects metadata and provide respectable learning material based on query request from the learner side. The proposed versioning model has to support multiple evolution scenarios: implementation change, interface change and registry change, Versioning learning object with metadata in e-learning system makes easy data search, retrieval and quick data delivery to learner.

This research article is organized as follows: A survey was taken on service versioning model is and explained in section 2. Section 3 describes the technical background of required for this paper. The proposed versioning model for personalized e-learning is presented in section 4. Section 5 shows the results and related works of this paper and finally section 6 concludes this paper.

2. Survey of Versioning Models

This section explains various versioning models. They have been classified into different categories that includes versioning model for enterprise, web service, learning objects, course generation, Adaptive Hypermedia and e-learning objects.

Rainer Weinreich [5] presented a versioning model for enterprise services within an SOA in the banking domain. Though approach has been developed for an SOA in a particular domain, it is not specific to this domain. A versioning model in the previously described context typically has to support three evolution scenarios: implementation change without changing the interface, compatible interface change, and incompatible interface change. The organizations in charge of existing Students need to be able to choose whether to continue using the new service with the old interface or whether to upgrade to be able to use the new functionality. The notification of Students is not part of the versioning model but part of the service governance process. From a technical perspective rules and guidelines for applying compatible changes to interfaces are
required.

David Frank [6] proposed two key concepts. The first is to draw a distinction between a service’s interface version and its implementation version. The second is to employ a tiered hosting model which echoes this structure, by introducing a version routing point in front of the actual service implementation versions. This version routing point routes versioned requests to the appropriate implementation versions.

Christopher Brooks [7] proposed the issues associated with creating derivative works based on learning objects in a general manner, and discusses the support that exists within current metadata specifications. Learning objects are reusable pieces of educational material intended to be strung together to form larger educational units such as activities, lessons, or whole courses. These materials are stored in learning object repositories which can be distributed in nature [14].

Tianxiang Lu [8] proposed a course generator provides personalized learning experiences by assembling structured sequences of learning objects that help learners to achieve their learning goals. They can be stored in a multitude of repositories and are selected by the course generator based on a set of pedagogical methods that take into account the learners’ goals and individual properties such as mastery. In this paper, we present the web-service we developed for the course generator of the LeActiveMath system. We describe the service oriented architecture during the design and the suitability via the application.

Evgeny Knutson [9] presents an approach that uses the terms, operations and methods of versioning applied to the Adaptive Hypermedia (AH) field. We show a number of examples of such a use which helps to facilitate authoring, managing, storing, maintenance, logging and analysis of AHS behavior, providing extensive flexibility and maintainability of the systems.

Wolfgang Theilmann, Michael Altenhofen [10] analyzed the requirements of e-learning systems for versioning support. Considering well-known design spaces for versioning systems we develop a versioning scheme that perfectly meets the requirement list. This so called implicit versioning scheme allows for transparent versioning and reuse of arbitrary content objects. We explicitly stress on the integration of the versioning schemes into the authoring process of e-learning material and discuss its effects on the usability.

Evgeny Knutov, Paul De Bra, Mykola Pechenizkiy [11] presented an approach that uses the terms, operations and methods of versioning applied to the Adaptive Hypermedia field. It showed a number of examples of such a use which helps to facilitate authoring, managing, storing, maintenance, logging and analysis of AHS behavior, providing extensive flexibility and maintainability of the systems.

3. Technological Background

This section describes the technological background information that includes SOA, service versioning, versioning model and personalized e-learning systems.

3.1. Service-Oriented Architecture

An SOA-based approach offers software components as a collection of distributed business services [12]. A business service plays the role of a single entity that represents a business application, and may include multiple IT resources, platforms, and components. The model separates the roles of consumer, provider, and registry. A service provider registers available services in a registry that the consumer can later discover and invoke. Service consumers aren’t directly aware of a specific service endpoint or the implementation details. The SOA provides a greater degree of separation between the provider and consumer, and as changes are made to a service, the SOA can help minimize the impact on the consumer.

One the most popular approaches for dealing with changes are versioning [29]. Versioning assumes simultaneous existence of multiple (different) implementations of the same thing, with every implementation distinguishable and individually addressable. In the case of SOA, service versioning equates to coexistence of multiple versions of the same service, which allows each consumer to use the version that it is designed for and tested for. In this case, a new version of a service is created based on the requirements of one or more consumers, which can start using this new version immediately. The other consumers of this service do not need to switch to using the latest version immediately, but can continue to use the versions of the service they were designed for and tested with. They can switch to the latest version of service, based on their own development and testing schedule.

This approach to versioning allows an IT organization to better adapt to changes required by the business. Multiple versions of a service can be maintained simultaneously without impact to the consumer. Old versions
can be retired gracefully, and consumers can upgrade on their own time, rather than being forced to when new versions are released.

3.2. Service Versioning Scenario

Service Versioning is a hot topic that has generated a broad range of guidance from a variety of sources. Service versioning scenarios are been grouped into two categories: message versioning and contract versioning [6]. Message versioning focuses on the data that composes the messages created and consumed by a service. Message versioning includes both forward and backward compatibility and extensibility [21]. Versioning permanently expands the base vocabulary for future implementations while extensions temporarily extend the base vocabulary for a specific implementation. Most message versioning requirements will fall into new documents, extending existing data constructs and message enhancements.

Service contracts should be designed with the assumption that once published, they remain relatively static [16]. Adopting this approach forces developers to build flexibility into their contract and schema designs, preferably by adopting a contract-first approach. While contracts should remain static, services need to evolve to remain useful and relevant to their consumers. It is possible to evolve a service in some ways that avoids breaking contracts with existing consumers. Most service contract requirements will fall into adding a new operation, adding new data structures and data types, adding a new interface for an existing operation, remember the first Tenet, type restriction in response messages, use <wsa:Action> and using a service broker.

There are several versioning scenarios for data contracts [15]. Possible changes are summarized in the following list as possible changes to a data contract and the affect it has on existing clients:

- Add new non-required members – Consumers unaffected. Missing values are initialized to defaults.
- Add new required members - An exception is thrown for missing values.
- Remove non-required members - Data lost at the service. Unable to return the full data set back to the Consumers, for example. No exceptions.
- Remove required members - An exception is thrown when Consumers receives responses from the service with missing values.
- Modify existing member data types - If types are compatible no exception but may receive unexpected results.

Versioning builds upon general XML versioning and extensibility guidelines. There are several common techniques for XML versioning and extensibility techniques available. They are namespaces Message, extension Elements and custom version attributes.

3.3. Versioning Strategies

Even though there is no de facto versioning technique for the WSDL, XML Schema, and WS-Policy content that comprise Web service contracts, a number of common and advocated versioning approaches have emerged, each with its own benefits and tradeoffs. The following are three known strategies [104]: Strict - Any compatible or incompatible changes result in a new version of the service contract. This approach does not support backwards or forwards compatibility. Flexible - Any incompatible change results in a new version of the service contract and the contract is designed to support backwards compatibility but not forwards compatibility. Loose - Any incompatible change results in a new version of the service contract and the contract is designed to support backwards compatibility and forwards compatibility.

A SOA inventory may be based on the already mentioned repository of service descriptions [38]. It may specify service versioning model, rules for new version announcement and service retirement process. Multiple coexisting versions of the same service in the system allows for the independent life cycles of services and their consumers and minimizes the overall impact of the introduction of changes. Although the necessity of such versioning mechanism may be obvious to anyone who has ever dealt with services, this topic still has not penetrated the mainstream of SOA publications and implementations.

Defining and publishing a "service versioning policy" that supports both the business need and create an efficient resource-utilization operating model. Support multiple versions where in business functionality may be altered based on the operating business context or business area or industry vertical. It allows an enterprise to reduce the risk to the business when introducing new business behavior or new business process optimization changes [28, 12]. Support multiple versions that support multiple versions of a service that allow the enterprise to incorporate and interject additional authorization, entitlement and audit related rules that change based on the
business context and/or business user role. Support multiple versions that allow the system administrators to provision server resources appropriately based on the QoS needs of the service consumer and the service contract definitions.

3.4. Personalized E-learning Systems

In personalized e-learning system Learning Objects are reusable pieces of educational material intended to be strung together to form larger educational units such as activities, lessons, or whole courses. These materials are stored in learning object repositories which can be distributed in nature [13]. Learning objects are small content components stored in the learning object repositories and are reusable and sharable in the different contexts and for different end users. Learning object repositories store both learning objects and their metadata either together or separately in order to facilitate their distribution and reuse. Learning object stored in centralized learning object repositories which consist of a single server to hold the learning objects and their metadata. Generally, learning object repositories can be classified into two major models of database. Centralized learning object repositories and distributed learning object repositories. The most common model is a centralized model where the learning objects are stored in a central database which resides on a single server for end-user to retrieve and assemble larger learning modules depending on individual learning needs [3].

3.5. Versioning of Learning Objects

In a centralized model learning objects are managed by a central database which resides on a single server. Communication between end users and the central database is carried out through a web service. This enables multiple users to retrieve and assemble learning objects easily, even if they are at different geographical locations. However, there are some drawbacks to applying the centralized database system. Centralized model has its limitations such as the data searching and retrieval time being delayed when the data volume is increasing continuously and the problem becoming worse if the database is not well designed. Figure 1 show the model of versioning learning objects. Applying Versioning model in learning objects which is stored in centralized database repositories, to support reuse existing learning object, and every reused learning object maintained as versioned learning object, versioned dynamically that is adding content, subtracting content, modifying content, reordering content and mentioned named for each version that make easy to search the data based on query from requester.

3.6. Metadata

Support for the versioning of learning objects comes through metadata specifications. The most mature and comprehensive metadata specifications for learning objects are the LOM [25] and the Dublin Core Metadata Initiative [18]. Both of these specifications contain support for capturing the fact that a relationship exists between two learning objects. It provides the details of the reuse learning object that contain versioned details and previous learning objects information. Comparing the both versioned learning object metadata and capture the suitable data and deliver based on the query request for service requester. Metadata is structured information system describing resources, created to help in the task of discovering, managing and using them without the need to be read, viewed or explored in some way [22]. This is the fundamental building block of the Semantic
Metadata enable effective search of resources across multiple repositories, since dealing with descriptive surrogates of resources is easier than dealing with the resources themselves. The use of a certain object by different communities can be facilitated by the existence of different metadata records describing it according to metadata schemes tailored to the needs of each community.

4. Service Versioning Model

This section describes the proposed service versioning model for personalized e-learning systems. The description includes motivation example, versioning model and the simulation.

4.1. Motivation Example

To serve the changing needs of the networked world, universities need to cooperate to reduce costs while responding to the growing demand. They have to specialize and develop their individual strengths to remain competitive. They must provide pervasive and personalized technologies and services. They need to integrate their abundance of learning resources and teaching competencies to compensate individual weaknesses and jointly succeed in the emerging international education market [17].

In e-learning system Learning Content Management System (LCMS) manages the learning content, eases content reusability, and provides workflow support during content development and delivery content via predefined interface and presentation layer [15, 16]. Learning objects are chosen as a component to develop an e-learning system. Reuse the arbitrary learning content is reusable learning objects. Its support adding content, subtracting content, modifying content, reordering content dynamically and managed by learning content management system [30]. Two key characteristic in the learning objects model are what it is used for and its reusability.

As the number of objects and authors grows, meta-data on the objects becomes a critical factor; indeed, meta-data is needed for an appropriate description of learning objects so that plug-and-play configuration of courses is enabled. Metadata in reality is data describing data and it can be used to describe any digital resource. There are various metadata elements which describe different aspects of digital resources. For example the IEEE LOM specification has metadata elements which enable the description of digital resources. By comparing two metadata and produce perfect adaptive learning material based on the service requester. Thus versioning model with metadata in e-learning system makes the function easy such as search, retrieval data and deliver to learner based on queried by requester. The implementation of proposed model of service versioning with different versions is shown in figure 1. A personalized e-learning consists of notification message, e-learning services with multiple versions version resolver, UDDI service registry and service updater. Applying a version model to maintain the reused learning objects as new version learning objects. Version model categorize every reusable learning objects this model make easy to search, retrieve data and deliver the data based on the query form service requester.

4.2. Backward Compatible

Backward Compatible means that newer versions of the service are compatible with earlier versions of itself. This ensures that clients written for old versions of the service can continue to operate with the new version as well. This can be achieved typically by preserving existing service contracts while extending the service by adding new features.

In the context of Web services, backward compatibility [23] is concerned with how changes to an interface affect existing service consumers of the interface. If existing service consumers are unaffected then the change is backward compatible. If existing service consumers are affected then the change is not backward compatible, and a strategy will be needed to manage the impact of the change. The concept of backward compatibility is closely related to the concepts of change and change management. It should be designed the Web services, there will be situations where the interface needs to change. The change needs to be made in a planned and systematic fashion, and it needs to be done according to a staged approach.

Many changes can be made to the interface that does not impact existing service consumers [24]. They are as follows:

- Adding a new operation: Existing service consumers will continue invoking existing operations while new operations become available for new consumers. Existing service consumers are unaffected.
- Adding new optional data structures to the input message: The existing service consumers are unaffected since they will be unaware of the new data structures.
Changing cardinality of existing input data structures from mandatory to optional: The existing service consumers continue using input data structures as if they were mandatory. Existing consumers are unaffected.

Change to the service provider implementation which has no material impact to the interface: Service implementation can be modified or totally rewritten as long as there is no material impact to the interface. That is, the existing interface specification is fully supported.

A range of changes will immediately "break" existing service consumers as are as follows:

- **Removal of an operation:** The old interface is no longer a proper subset of the new one. Existing service consumers using the removed operation will be impacted.

- **Changing cardinality of existing output data structures:** Change to the cardinality of fields in the output message, such as changing mandatory fields to optional fields, will break existing service consumers.

- **Change to the definition of data types:** Most changes to the data types in the input or output message are not backward compatible.

- **Business rules or business process changes:** Sometimes there is a dependency between Web service operations. This dependency typically arises because operations must be invoked in a prescribed order. Should the underlying business rules or business process change, the dependency changes and the existing clients no longer operate correctly even when the interface remains unchanged or is changed in a backward compatible manner.

### 4.3. Forward Compatible

Forward Compatible means that newer versions of the service can be deployed without breaking existing clients. The service can accommodate a number of unknown and unexpected features. Standards that have been developed to aid in ignoring unrecognizable elements that appear in the future have been an aid in building forward compatibility.

Two more concepts that are associated with versioning are extensibility and graceful degradation. **Extensibility** is an architectural principle that enables extensions to be implemented without impacting current implementations. It may require adding new functionality or modifying existing functionality. It is one of the key features of XML. **Graceful degradation** describes service behavior in unexpected circumstances. Erroneous
and unexpected exceptions must be dealt with using a safe and appropriate response. It applies both to service behavior and to messages used by the service.

The managing service evolution using compatible XML schema design approach is presented in fig2. For compatible evolution the schema of the request messages has to change in a backward compatible manner and the response has to change in a forward compatible manner. Here the strategy is based on the service provider WSDL.

The managing service evolution using service adapter approach is explained in fig3. It presents how an older request message from V 1.0 requester transformed into V 1.1 request message. The opposite is done for the response message where V 1.1 from the service transformed into a V 1.0 response message. The Service evolution using multiple service version approach is explained in figure 4. It maintains multiple versions of the service and configures the requesters to talk to the appropriate service versions using content routing algorithm.

4.4. Proposed Versioning Model

The implementation of proposed model of service versioning with different versions is shown in fig 5. A personalized e-learning consists of notification message, e-learning services with multiple versions version resolver, UDDI service registry and service updater.

The learners use either via web service protocols and a web service endpoint in case of web services. Web services are addressed via URL containing the service name, the name of the subsystem’s versions number. The learner can login/logout, view content, uploads assignment to the server. Each learner should register themselves as a client. The notification message uses a version identifier. This version identifier identifies the version of the WSDL documents that the calling component used to call a particular service. There is always a WSDL document for every service versions. The e-learning services with different versions are registered in a UDDI service registry. For every version of the service the registry holds the version identifier of the corresponding WSDL documents. In addition, for each registration of a version service the registry holds a backward compatibility with every newly registered versions of a service a list of compatible earlier versions is registered.

The service updater keeps on monitor the UDDI service registry. Whenever adding a new service, or deleting a service or updating service, the service updater gets the subscriber list from the application and sends the
message to all the registered clients. Upon receiving a notification message from publisher, the application server forwards the notification to the relevant client subscribers by comparing the message content with the list of subscriptions it holds. It provides the flexibility to operate in a dynamic environment and is independent of the need to configure information relating to the clients of a notification. The notification itself is encapsulated within an object and contains a list of key value pairs. The service updater expresses its interest with a subscription element, which is built with a special subscription language containing simple logical expressions.

4.5. Service Description

The service specification defines the and the how for a service interaction, but it is not enough on its own to truly enable an interaction. In particular, there is no information that indicates the location of a particular instance of a service provider. The entity called the service description contains all the information needed to interact with an instance of a version of a service. The service description is made up of (by inclusion or reference) the service specification to define relevant external functional and non-functional characteristics (the what and how), and a service instance address (the where).

The service description itself has a unique identifier of the same form as a service specification. In fact, the service description identifier reflects the service specification identifier, but the version adds a micro to the service specification version to account for maintenance changes. The micro increments when internal changes are made that can affect the tacit behavior relevant to a particular version of a service specification. Examples include different configuration files or rules, and even bug fixes in the implementation or assembly for the service instance. A service description can include various forms of tags to distinguish one service instance from others enacting the same service specification. The service description is typically published (as metadata) in a Service Registry. In general, the associated service specification is published (as metadata) as well.

4.6. Versioning Metadata

The service version aggregates all service instances that implement the same version of a service specification for a service. The service version comprises (by inclusion or reference) one version of a service specification and one or more service descriptions. The service version itself has a unique identifier that is the same as the service specification identifier. The service version, however, is not itself a versioned entity in the model; it simply represents all instances of a particular service version. Like the other entities, the service version is typically published (as metadata) in a service registry.

Current service registries such as UDDI do not provide direct support for different versions of the same service. For each service a set of metadata and a service version is stored. Versioning metadata (i.e., the relationships in the service version graph) are defined by the service provider when publishing new services or revisions. It is the service provider’s decision whether the differences between two service versions are so fundamental that an entire new service should be created instead. However, every evolutionary change mandatorily leads to the creation of either a new revision or a new service. It is not possible to just “update” the
4.7. Evaluating the Proposed Model

The proposed model was evaluated by portability, deployability, scalability and usability aspects. Good portability is one strong point in the proposed model. Even so, the architecture browser based service is forced to select between many alternative technologies that are largely incompatible with each other. In principle the deployability of a browser based service is as simple as it gets. Any user with a browser that is compatible with the service can start using the service immediately. A registration function can be taken inside the browser session. The scalability in capacity of browser based architecture can be controlled by the traditional design solutions. The scalability may be access to the service and the communication between the service components in the network. Usability of an individual service can be reasonably good for suitable applications. Typically the interaction is taken between the user and the application.

5. Results and Related Works

Support of multiple concurrent operational versions of a service leads more flexible and robust systems. They make use of the highest available compatible version of the service on the fly without any modification. Multiple incompatible versions of a service may exist concurrently without any effect on the calling components. New versions of a service can easily be tested in a production environment, without disturbing the current version. Versions of which higher compatible versions are available - won't be called anymore and can be removed without any modifications to the calling components. The evolution of Web services is subject to a wide ranging debate. The World Wide Web Consortium recently paid attention to versioning of Web services. Currently, a common workaround to deal with the lack of versioning support is to use separate namespaces for each version of a Web service. Some of the works that related to this paper are version related extensions to variables, WSDL, service interface, version aware service model.

Matjaz B. Juric, Ana Sasa, et. al. [19] addressed version-related extensions to variables and introduces version handlers. The proposed extensions represent a complete solution for process-level and scope-level versioning at development, deployment, and run-time. It also provides means to version applications that consist of several BPEL processes, and to put temporal constraints on versions. The proposed approach has been tested in real-world environment. It solves major challenges in BPEL versioning.

David Frank, Linh Lam et.al. [20] presented an approach, which leverages the existing WSDL service definition model to build a versioned service hosting solution. It distinguished between a web service interfaces (published) version and its implementation version (private). They introduced the concept of a service interface proxy.

David Frank, Linh Lam et.al. [6] explained two key concepts. The first is to draw a distinction between a service’s interface version and its implementation version. The second is to employ a tiered hosting model which echoes this structure, by introducing a version routing point in front of the actual service implementation versions. This version routing point routes versioned requests to the appropriate implementation versions. They have implemented a prototype of their approach on IBM’s Web Sphere Process Server and Web Sphere Enterprise Service Bus. It showed results with a versioned test service to demonstrate the capabilities of the proposal.

Ru Fang, Linh Lam, Liana Fong et.al. [22] introduced a version-aware service model based on some architectural extensions to WSDL and UDDI. WSDL would be enhanced to describe the attributes of the service versions. UDDI would be augmented to use versions in a service directory with an event-based notification/subscription mechanism. They designed a proxy, residing in the service consumer side which can dynamically update the client application instance at runtime. We have implemented a prototype to demonstrate these models and used a weather forecast web service as an example to illustrate the usefulness of the proposed architecture.

6. Conclusion and Future Enhancements

This paper discussed service-oriented architecture, web services and e-learning systems. Service versioning model overcomes the drawbacks such as data searching and retrieval time being delayed when the data volume is increasing continuously and the problem becoming worse if the learning content management system is not
well designed. In terms of future work will included efficient metadata with instructional management specifications for search and retrieve data from the different group of services and add huge amount data in continues. This will add the versioned learning object of analysis of quality of the courses, through the development of a standardization metadata to import external courses.

Acknowledgments

We would like to thank A. Tamilselvan, Department of Computer Science, School of Engineering, Pondicherry University, Puducherry – 605014, India for constant supporting in this research work. We are grateful to our respected Marie Stanislas Ashok, Systems Manager, Computer Centre of Pondicherry University, Puducherry – 605014, India for providing us with all the facility that were required in this research work.

References

[8] Tianxiang Lu; Carsten Ullrich; Babara Grabowski; (2008): Course Generation as a Web-Service (CGWS) for E-Learning Systems, German Research Center for Artificial Intelligence Saarbrucken, Germany.
[20] David Frank; Linh Lam; Liana Fong; Ru Fang; Manoj Khangaonkar; (2008): Using an Interface Proxy to Host Versioned Web Services, IEEE International Conference on Services Computing.
[22] Ru Fang; Linh Lam; Liana Fong; David Frank; Christopher Vignola; Ying Chen; Nan Du; (2007): A Version-aware Approach for Web Service Directory, IEEE International Conference on Web Services (ICWS 2007).
[29] Sathiamoorthy; Manoharan; (2004): Application State, Serialization and Versioning, IASTED International Conference on Advances in Computer Science and Technology.