British Journal of Mathematics & Computer Science
10(1): 1-9, 2015, Article no.BJMCS.18505
ISSN: 2231-0851

SCIENCEDOMAIN *international*
www.sciencedomain.org

A New Layer of Service Oriented Architecture to Solve Versioning Problem

Soheir Zein[1], Kassem Akil[2], Ali Kalakech[2] and Seifedine Kadry[3*]

[1]Arts, Sciences and Technology University, Beirut, Lebanon.
[2]Faculty of Business Administration, Lebanese University, Lebanon.
[3]School of Engineering, American University of the Middle East, Kuwait.

Article Information

DOI: 10.9734/BJMCS/2015/18505
Editor(s):
(1) Qiang Duan, Information Sciences & Technology Department, The Pennsylvania State University, USA.
(2) Tian-Xiao He, Department of Mathematics and Computer Science, Illinois Wesleyan University, USA.
Reviewers:
(1) R. S. Ponmagal, Dept. of CSE, Dr. M. G. R. Educational and Research Institute University, Chennai, India.
(2) Shrikant Upadhyay, Electronics & Communication Engg. Dept. Cambridge Institute of Technology, Ranchi, India.
(3) Anonymous, Mansoura University, Egypt.
(4) Anonymous, Kocaeli University, Turkey.
Complete Peer review History: http://sciencedomain.org/review-history/9853

Original Research Article

Received: 26 April 2015
Accepted: 05 June 2015
Published: 18 June 2015

Abstract

Aims: Service Oriented Architecture (SOA) is an architecture where reusable solution is provided to solve the common problem in software design. The major advantage of SOA is the light dependent between its functions and services. SOA services are defined through description language, independent of any vendor or technology. Therefore, developers can use SOA services in a standard way without need to know or understand how functions of a service are implemented. SOA services need to evolve to meet business changes and requirements. Even though the light independency of SOA services, any change or update to the implementation of a service will impact other parties who use this service. So SOA versioning service becomes a strong issue. In this paper we will discuss SOA Layers and service versioning problem. We will propose a new layer to SOA architecture to handles service versioning issue and the structure of the new layer will be presented.

Keywords: Service oriented architecture; service versioning; SOA layers.

1 Introduction

Service-orientation needs loose coupling of services with operating systems, and other technologies that are the basis of applications. SOA divides functions into separate units, or services, which developers make it accessible over a network to enable users to merge and reuse them in manufacturing of applications. These

**Corresponding author: skadry@gmail.com;*

services and their respective consumers communicate by passing data in a well-defined, shared format, or by coordinating an activity between two or more services.

SOA implementations exist in almost all industries, but SOA fails when wrong decisions are taken or when there is lack of knowledge within the enterprise deploying SOA. A good governance is needed to ensure that all of the operations of a Service come together to meet the enterprise SOA requirements.

The most SOA burning issue is service versioning. After manufacturing new services into production and start using them, the users of those services start needing changes or updates to these services. Also, bugs need to be solved, interfaces need to be redesigned, adding new functionalities is needed, and unnecessary functions need to be removed. Every change introduces a new version of the service. Service versioning facilitates running the system stable without interruption to existing proven service functions and lets you add new functionalities to those services.

With the existence of the service consumers, changes need to be made without disturbing them and the successfulness of their operations. At the same time, services need to meet the needs of users with new requirements. Service versioning must meet these goals. Version compatibility enables the consumers to invoke different but compatible version of the desired service.

In this paper we will present a new solution to of service versioning issue by adding a layer to the SOA layers to cover and handle major service lifecycle issues. We will present the related work in section 2, SOA layers in section 3. In section 4 we discuss and suggest adding a layer to the SOA layers with a message router that uses routing mechanism and show how it works. We will present versioning layer with a message router 5. The paper is concluded with a summary of the practices that are covered in section 6.

2 Related Work

Many companies and researchers have proposed governance methods to handle versioning of services. The scope and coverage of these methods differ extensively. Nevertheless, these methods do not completely cover all important elements of service lifecycle management.

In [1-3] the author distinguishes between two kinds of service changes shallow versus deep service changes. With shallow changes the change effects are localized to a service or are strictly restricted to the clients of that service.

Shallow changes characterize both singular services and business processes, and require a structured approach and robust versioning strategy to support multiple versions of services and business protocols. To deal with shallow changes he introduced a theoretical approach for structural service changes focusing on service compatibility, compliance, conformance, and substitutability. In addition, he described versioning mechanisms for handling business protocol changes. The right versioning strategy can maximize code reuse and provide a more manageable approach to the deployment and maintenance of services and protocols. It can allow for upgrades and improvements to be made to a service or protocol, while supporting previously released versions.

In [4-9], the authors have provided guidance on the typical issues that a versioning policy for a service oriented system must address. They also gave specific guidance on implementing and supporting versioning capabilities in technologies that are commonly used for building services. Finally, they highlighted common problems that can occur in a poorly versioned service-oriented system to reinforce the need for comprehensive versioning policies. They also summarize their recommendations.

In [10,11], the authors presented a general approach to versioning of service-oriented systems using service version graphs and selection strategies, and demonstrated how this approach is implemented in their service runtime environment. Their approach leverages the dynamic invocation mechanism of service that enables to

invoke different versions using transparently rebinding service proxies. They also illustrated the advantages of their approach in comparison to UDDI (Universal Description, Discovery and Integration) and Apache WSIF (Web Services Invocation Framework) using a case study from the telecommunications domain.

In [12], authors identify the types of changes that can occur in services, introduce different patterns that can be considered for Web services versioning, and provides guidelines for applying those patterns in real-world solutions. The releases of services are classified to major and minor releases, then three patterns are introduced and solutions for applying these patterns are provided using Oracle Service Bus.

3 SOA Layer

SOA architecture enables the integration of existing applications by exposing their functionality as services to improve the value of existing software assets and avoiding redundancy in IT infrastructure. To yield these benefits and to support the service-oriented paradigm, SOA adds further layers to the conventional architecture responsible for application integration and service orchestration hence making the underlying systems transparent for business analysts [13]. A reference model of SOA layering is provided in [13] which introduce three basic layers of SOA the Integration layer, the Process layer, and the presentation layer.

Here is Fig. 1 that illustrates the SOA layers and how they are positioned on the top of each other.

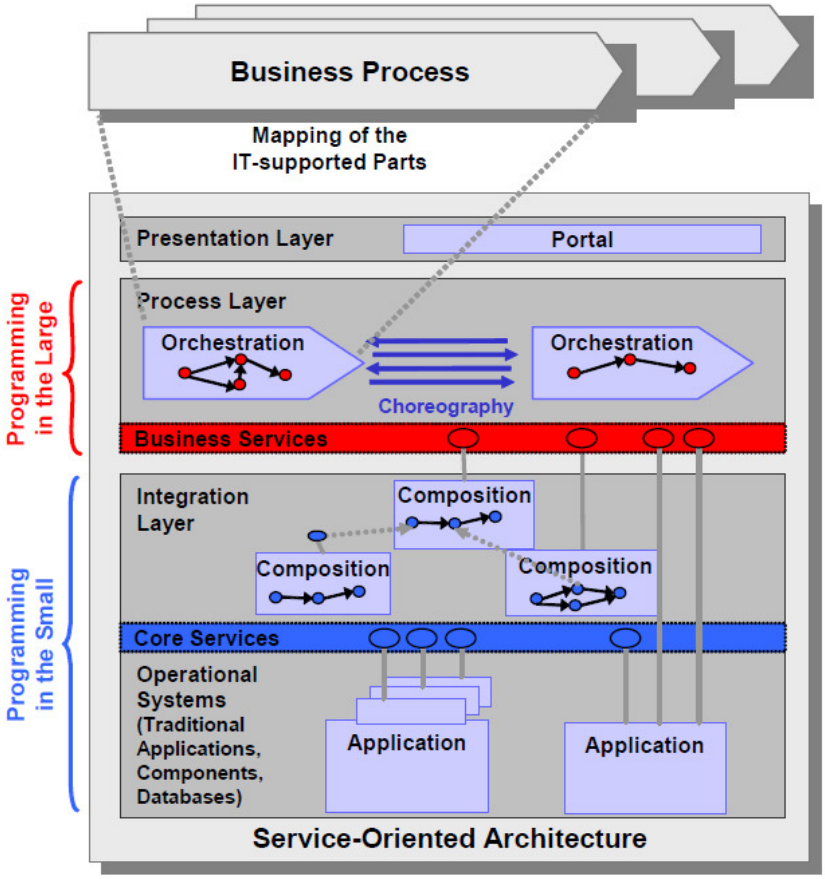

Fig. 1. SOA layers

4 New Proposed SOA Layer

The issue of service development and change management is a complex one, and this research does not attempt to involve every aspect surrounding the evolution of services. However, it introduces some key approaches and helpful practices that can be used as a springboard for any further research in service evolution.

The proposed solution is obtained by adding a layer to the SOA layers to cover and support the SOA principles and requirements when releasing a new service and during moving from service to another service.

SOA is not a ready technique, but a method to build and organize the IT basic structures and service functions, and a method to design, develop, deploy, and manage discrete logic units (services) in a specified environment in an organized way to handle major service lifecycle issues. Recall that typically SOA has three layers. Each layer is responsible of a set of responsibilities.

All the above leads us to a defined conclusion that, to handle service government during the versioning process, you can add a versioning layer between process layer and integration layer and turn it to four SOA layers. The four layers are: presentation layer, process layer, versioning layer, and integration layer. Each layer depends on the lower layer for implementation and provides interface for the upper layer. The new proposed structure will be as Fig. 2:

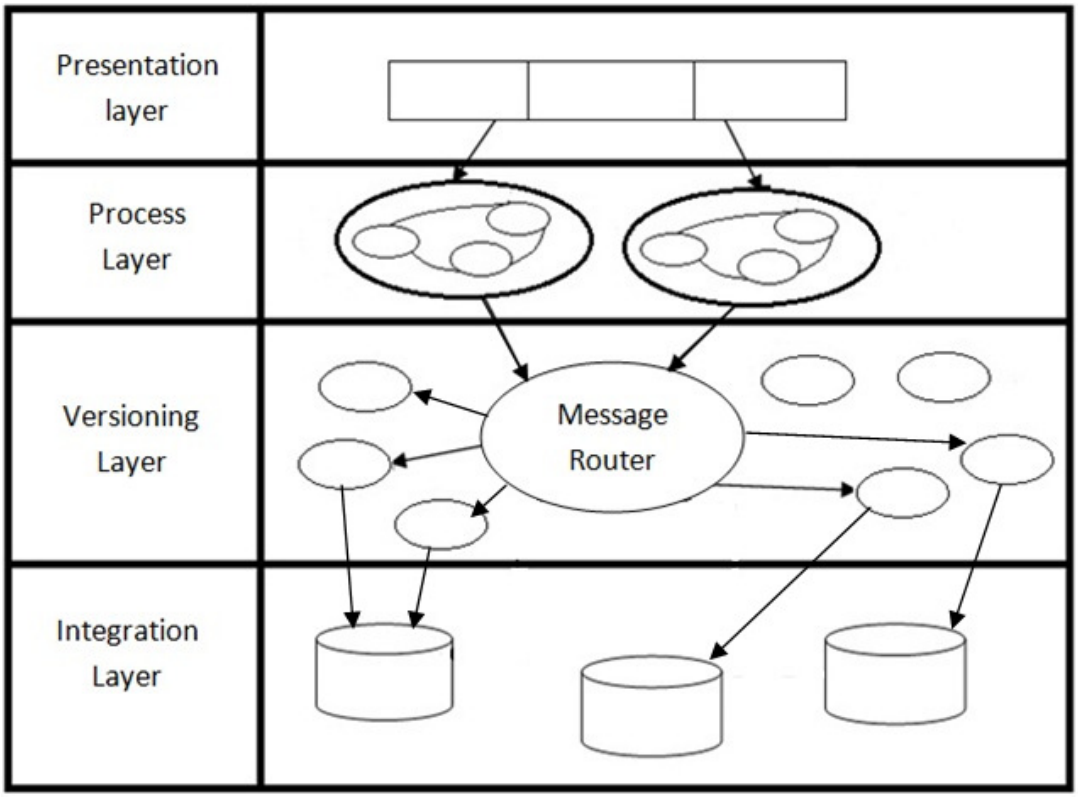

Fig. 2. New structure of SOA layers

The figure above shows the four layers:

1) Presentation layer: implements man-machine interaction and mainly deals with customers' requests and displays system information. It may consist of: portals, terminals, client apps, and web pages.
2) Process layer: deals with flows of programs and handles complex business logics in the system and composes the business logic function by using multiple business processes. Its tasks: workflows, orchestration, and choreography.
3) Versioning layer: handles the service versioning processes and the rebinding of the service consumers to the right version of a service. It covers all aspects of the service life-cycle management, it provides the ability to mediate, aggregate, split, transform, route and transport service requests reliably from the service requester to the service provider.
4) Integration layer: handles the database and stores enterprise data. It may contain: data stores, file systems, data access, and the existing software systems.

5. Versioning Layer with a Message Router

When a breaking change is made to a service then a new version of a service is released, the consumer must migrate to the new release. This capability is provided by the versioning layer through a router that uses a routing mechanism through content-based routing to route messages to the appropriate version of a service.

The existence of the router in the versioning layer permits the coexisting of two releases of a service without changing the consumers' code, and helps to guarantee a refined migration to a new release.

In case of non-breaking change the service implementation must be required to support different versions of its interface simultaneously, this is because not all clients will upgrade to the new version. The router can examine the message and depending on the presence of certain elements of that version, routes the message to the other version of the same interface.

The core idea besides forwarding messages to a client endpoint, the ability to route the messages targeted to version1 to version 2 of the service, this routing can be done by many ways depending on the services programming standards. In this research the message routing mechanism route messages based on message content of a SOAP-based messages that is found in the message head indicating to which service version the message will be forwarded.

The router receives messages and then routes each message to the requested services based on the message content. The router relies on methods used to match the values within the message with the values it has. If it matches, the message is routed to the requested service. If not, then it routes the message to the compatible service. Values are grouped in a router tables. The router tables contain all the services and their compatible services, so if a service is removed the message will be routed to the compatible one. The routing Table 1 must be updated permanently. This can allow the existence of many versions of a service and also for routing users to the appropriate services.

The simplest case is the non-breaking changes where a non-functional requirement is changed that can be a correction of a bug or a performance improvement (Fig. 3). This change will not have an effect on the Web Services Description Language (WSDL) of the Web service and the service interface. The version number of a service would be for example (v 1.00), so after the changes it will be (v1.01). This type of change is transparent and the router can send the request to the new version without notifying the consumer because the interface is not changed so the router cares only about (v1.0) from the message.

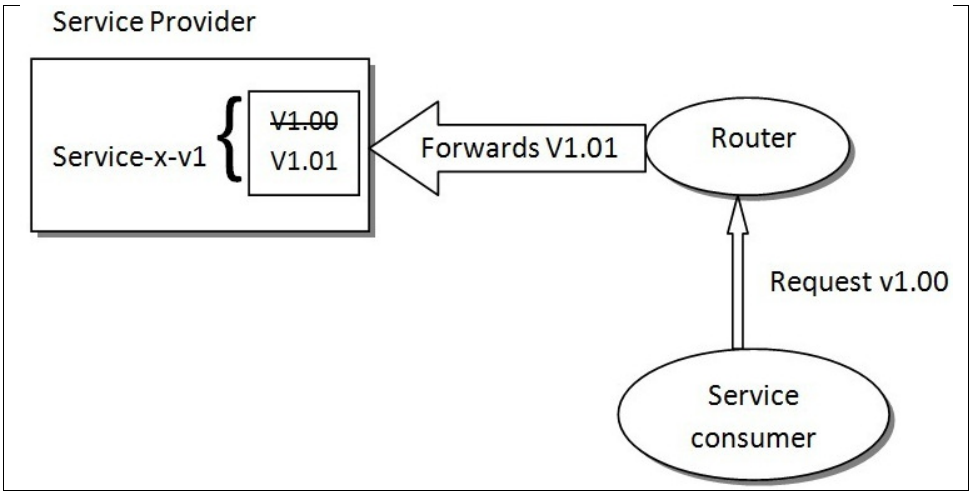

Fig. 3. Non-functional of non-breaking change

The second case is in the non-breaking changes, where new optional elements, operations, or attributes are added to existing service (Fig. 4). In this case the version number will be (v1.10). Since this case provides a compatible interface, the consumer must be notified and they can choose to stay using the old version or switching to the new one. If the service changes constantly, the router will have a record about the number of versions released, then it compare the number of versions with the allowed number of versions that is set by the service provider, if it is greater than the number allowed, it uses a replacement method to switch the users of the oldest version of the service to a newer one and then remove the oldest. So, when a user of the oldest version sends a message the router will forward the message depending on the version number that is before the dot (v1) only and neglect the rest of the number and choose the version that was released after the oldest one.

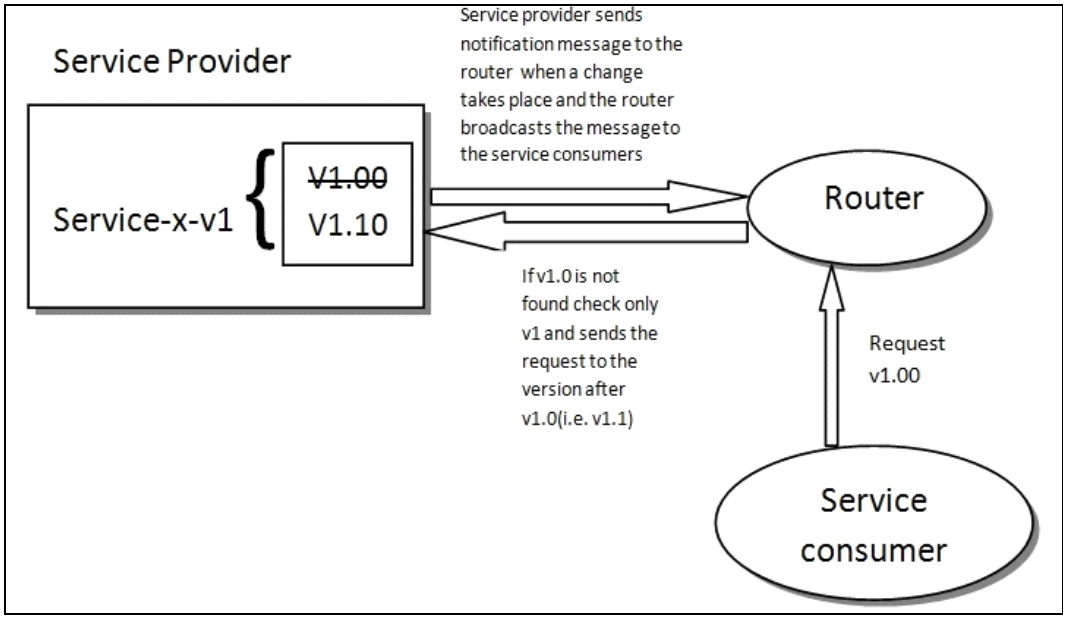

Fig. 4. Non-breaking functional change

In both cases mentioned above, if the service consumer wants to use the newest version every time it is released then he can send only the first part of version number (i.e. v1) and the router will send the request always to the newest version of the service by default.

The third case is where a breaking change occurs (Fig. 5). In this case, removing or replacing of operation, attributes, or required elements from a service takes place and the service consumers must upgrade to the newer version. During this process the service provider must mark the old versions as deprecated and send messages to the service consumers that the service has a newer version and it is in the grace period so they must move to the newer version. After making sure that all the consumers have moved over, remove the old one. Removing the service effectively should be planned and announced ahead of time. This can be done only when all the service consumers are known by the service provider. Otherwise the service must not be removed until it is been never used. And what if some consumers didn't move and didn't update their codes? The version number in this case will be (v2.00) and the router must forward the message to the appropriate service version by routing it to the most recent or most compatible version of the service. So when a consumer sends a message that has the version number (v1.0 or v1.1 or v1) the router will match it in the router table, if it will not find the version of that service it will increment the version number and search for next version and then send the message to the newer version that it will find.

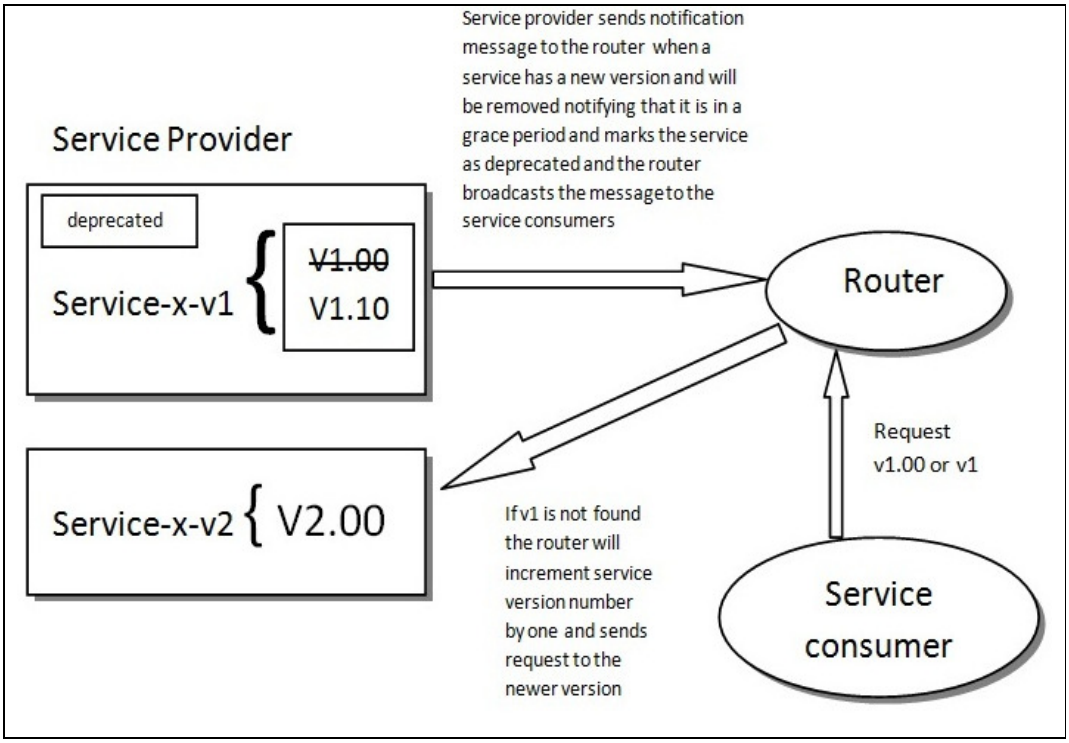

Fig. 5. Breaking change

Table 1. Routing table

Service name	Version	Compatible services	Active	Default	Location	Description
X	V1.00	V1.01- V1.10	No	V1.01	…….	…….
X	V1.01	V1.10	Yes	V1.01	…….	…….
X	V1.10	V2	Yes	V1.10	…….	…….

6 Conclusion

According to the above analysis, we can see that SOA environment is defined in terms of a service.

In summary, a service-oriented business environment in accordance with the principles of SOA will fail without an intelligent or exact SOA Governance. Seeing that, the encouragement underlying this study can be explained by the fact representing that a high percentage of all SOA developmental hard work fail. There are a lot of causes behind these failures. One of the aspects of SOA governance is service versioning. Soon, when a service is made available the users of those services begin needing changes, the bugs need to be fixed, maybe a new functionality is added, or the interfaces are redesigned, and the unneeded functionalities are removed. With the presence of the service consumers, all the changes need to be made without annoying them and the successfulness of their processes. Also, services need to adapt the needs of users with new requirements. Service versioning have to meet these goals. Version compatibility allows the consumers to invoke unlike but compatible version of the preferred service.

Service versioning can be handled by several ways, depending on many factors and constraints to absorb a lot of misunderstanding to this process. In this research we have covered several practices that can be done in versioning task in order to:

- Deploy more than one version of a service at the same time.
- Access several versions of a service by the same consumer or different consumers.
- Routing messages, in case of breaking or non-breaking versioning, targeted to old versions to the newer versions.
- Deprecating then removing old services in an organized manner.

Competing Interests

Authors have declared that no competing interests exist.

References

[1] Mike P. Papazoglou. The Challenges of Service Evolution. 20th International Conference, CAiSE 2008 Montpellier, France; 2008

[2] Bin Sarib AS, Haifeng Shen. SORC: Service-oriented distributed revision control for collaborative web programming. Computer Supported Cooperative Work in Design (CSCWD), Proceedings of the 2014 IEEE 18th International Conference. 2014;638-643.

[3] Leitner P, Michlmayr A, Rosenberg F, Dustdar S. End-to-End versioning support for web services. Services Computing, 2008. SCC '08. IEEE International Conference. 2008;1:59-66.

[4] Bechara G. Web services versioning; 2007. [Online].
Available: http: //www.oracle.com/technology/pub/articles/web services versioning.html

[5] Christian Emig, Kim Langer, Karsten Krutz, Stefan Link, Christof Momm, Sebastian Abeck. The SOA's Layers. C&M Research Report, Karlsruhe; 2006.

[6] Seifedine Kadry. An Implementation of ODBC Web Service. Journal of Theoretical and Applied Information Technology. 2011;26(1).

[7] Aleksander Dikanski, Sebastian Abeck. A View-based Approach for Service-oriented security architecture specification. The Sixth International Conference on Internet and Web Applications and Services (ICIW 2011), St. Maarten, Netherlands Antilles, März; 2011.

[8] Kadry Seifedine, Smaili Khaled. A solutions for authentication of web service users. Information Technology Journal. 2007;6(7):987.

[9] Hazzaa F, Kadry S. New system of E-voting using fingerprint. International Journal of Emerging Technology and Advanced Engineering. 2012;2:355-363.

[10] Barry D. Web services and service-oriented architectures; 2004.
Available: http://www.service-architecture.com onApril 14, 2006.

[11] Ingo Pansa, Felix Palmen, Sebstian Abeck, Klaus Scheibenberger. A domain-driven approach for designing management services. The Second International Conferences on Advanced Service Computing (SERVICE COMPUTATION), Lissabon, Portugal; 2010.

[12] Hongqi Li, Zhuang Wu. Research on distributed architecture based on SOA, communication software and networks, 2009. ICCSN '09. International Conference on. 2009;670-674.

[13] Fernández Erica, Toledo Carlos M, Galli María R, Salomone Enrique, Chiotti Omar. Agent-based monitoring service for management of disruptive events in supply chains. By. In Computers in Industry. 2015;70:89-101. Language: English. DOI: 10.1016/j.compind.2015.01.009.

Peer-review history:
The peer review history for this paper can be accessed here (Please copy paste the total link in your browser address bar)
http://sciencedomain.org/review-history/9853