

Original Research Article

A New Layer of Service Oriented Architecture to Solve Versioning Problem

ABSTRACT

Aims: Service Oriented Architecture presents flexibility and capability for application developers due largely to the fact that SOA components are loosely coupled and exposed as independent services on a network. Therefore, application developers can access SOA services in a standardized way and without understanding how the service is implemented. This revolutionary concept in software architecture is popular because SOA reduces development time, promotes software reuse, and reduces project costs. The most SOA burning issue is service versioning. When a service is made available, soon, the users of those services start needing changes, bugs need to be fixed, new functionality added, interfaces redesigned, and unneeded functionality removed. In this paper we will discuss SOA Layers and service versioning. Once the detailed mentioned topics are done, we will propose adding a layer for handling service versioning and the structure of the new layers will be presented.

Keywords: *Service Oriented Architecture, Service Versioning, SOA Layers, Change Management, Message Routing.*

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 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 SOA layers in section 2. In section 3 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 the related work in section 4. The paper is concluded with a summary of the practices that are covered in section 5.

2. SOA LAYER

SOA enables the integration of existing applications by exposing their functionality as services improving 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 [1]. Opinions about SOA layering are disjunctive and contradictory within the SOA community and no standardized layering is established. A reference model of SOA layering is provided in [1] which introduces three basic layers of SOA the Integration layer, the Process layer, and the presentation layer. Here is figure-1 that illustrates the SOA layers and how they are positioned on the top of each other.

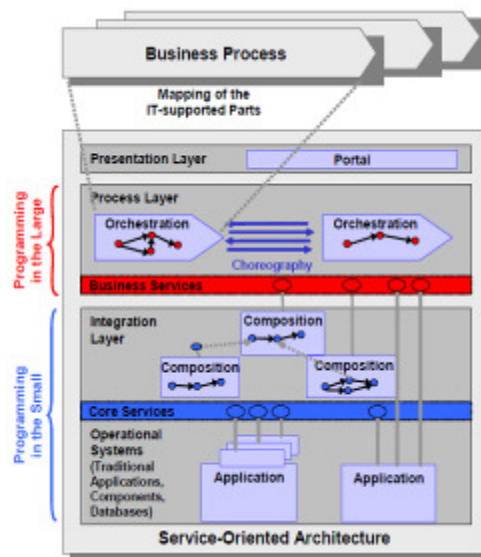


Figure-1- SOA layers

3. NEW PROPOSED SOLUTION

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 structure will be as follows:

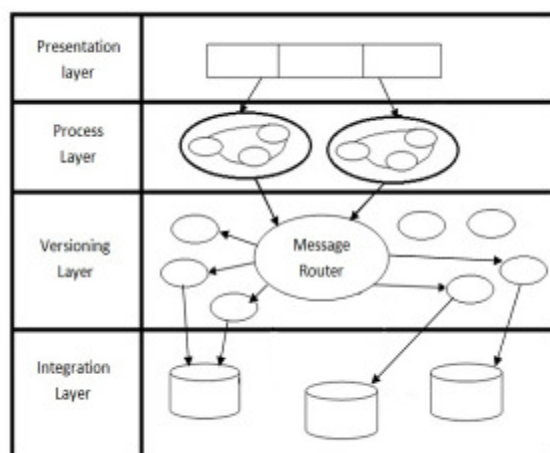


Figure -2- New Structure of SOA layers

The figure above shows the four layers:

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.

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.

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.

Integration layer: handles the database and stores enterprise data. It may contain: data stores, file systems, data access, and the existing software systems.

4. 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.

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

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

116
117 The core idea besides forwarding messages to a client endpoint, the ability to route the
118 messages targeted to version1 to version2 of the service, this routing can be done by many
119 ways depending on the services programming standards. In this research the message
120 routing mechanism route messages based on message content of a SOAP-based messages
121 that is found in the message head indicating to which service version the message will be
122 forwarded.

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

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

138

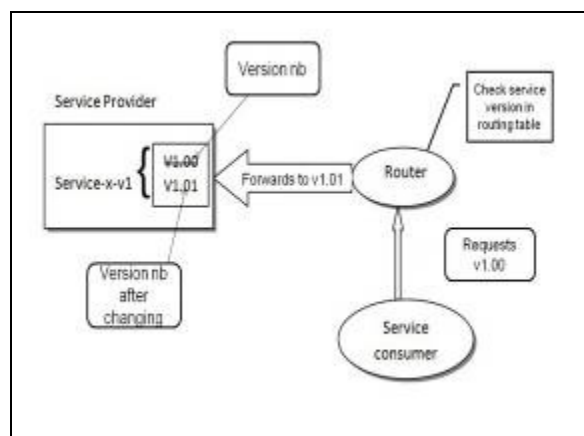


Figure 8: Non-functional of Non-breaking change

139

140

141

142 The second case in the non-breaking changes where new optional elements, operations, or
143 attributes are added to existing service. In this case the version number will be (v1.10) .
144 Since this case provides a compatible interface, the consumer must be notified and they can
145 choose to stay using the old version or switching to the new one. If the service changes
146 constantly, the router will have a record about the number of versions released, then it
147 compare the number of versions with the allowed number of versions that is set by the

148 service provider, if it is greater than the number allowed, it uses a replacement method to
 149 switch the users of the oldest version of the service to a newer one and then remove the
 150 oldest. So, when a user of the oldest version sends a message the router will forward the
 151 message depending on the version number that is before the dot (v1) only and neglect the
 152 rest of the number and choose the version that was released after the oldest one.
 153

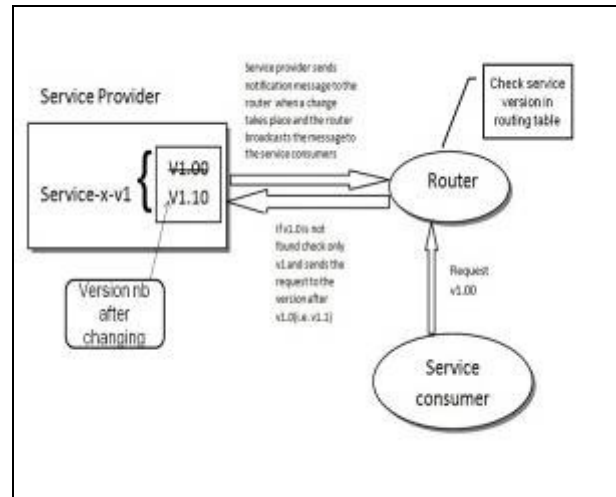


Figure 9: Non-breaking Functional change

154
 155
 156
 157 In both cases mentioned above, if the service consumer wants to use the newest version
 158 every time it is released then he can send only the first part of version number (i.e. v1) and
 159 the router will send the request always to the newest version of the service by default.
 160 The third case is where a breaking change occurs. In this case, removing or replacing of
 161 operation, attributes, or required elements from a service takes place and the service
 162 consumers must upgrade to the newer version. During this process the service provider
 163 must mark the old versions as deprecated and send messages to the service consumers
 164 that the service has a newer version and it is in the grace period so they must move to the
 165 newer version. After making sure that all the consumers have moved over, remove the old
 166 one. Removing the service effectively should be planned and announced ahead of time. This
 167 can be done only when all the service consumers are known by the service provider.
 168 Otherwise the service must not be removed until it is been never used. And what if some
 169 consumers didn't move and didn't update their codes? The version number in this case will
 170 be (v2.00) and the router must forward the message to the appropriate service version by
 171 routing it to the most recent or most compatible version of the service. So when a consumer
 172 sends a message that has the version number (v1.0 or v1.1 or v1) the router will match it in
 173 the router table, if it will not find the version of that service it will increment the version
 174 number and search for next version and then send the message to the newer version that it
 175 will find.
 176

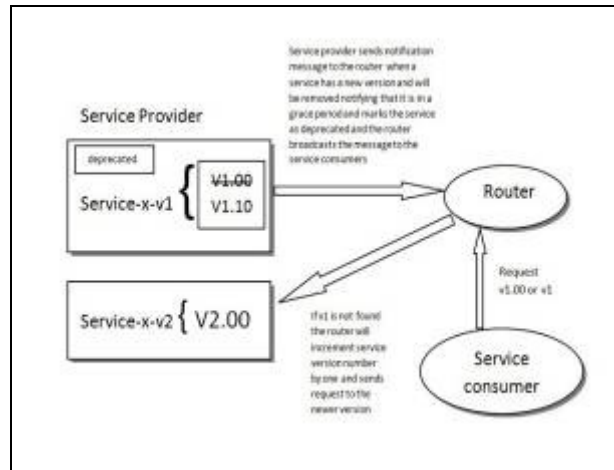


Figure 10: Breaking change

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	V1.01	V1.10	No	V1.10	-----	-----
X	V1.10	V2	Yes	V1.10	-----	-----
X	V1.10	V2	No	V2	-----	-----

FIGURE 11: ROUTING TABLE

5. 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 [14] 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. Deep changes cause cascading types of changes which extend beyond the clients of a service possibly to entire value-chain, i.e., clients of these service clients such as outsourcers or suppliers. Typical shallow changes are changes on the structural level and business protocol changes, while typical deep changes include operational behavior changes and policy induced changes.

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.

Deep changes characterize only business processes and require that a business process be redefined and realigned within an entire business process value chain. This may eventually lead to modification and alignment of business processes within a business process value

chain associated directly or indirectly with a business-process-in-scope. To address these problems he introduced a change-oriented service life cycle methodology. A change-oriented service life cycle provides a sound foundation for deep service changes in an orderly fashion that allow services to be appropriately (re)-configured, aligned and controlled as changes occur. A change-oriented service life cycle also provides common tools to reduce cost minimize risk exposure and improve development agility. It helps organizations ensure that the right versions of the right processes are available at all times, and that they can provide an audit trail of changes across the service lifecycle to prevent application failures and help meet increasingly stringent regulatory requirements.

In [15], 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 [16], 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 VRESCO service runtime environment. Their approach leverages the dynamic invocation mechanism of VRESCO that enables to invoke different versions using transparently rebinding service proxies. They also illustrated the advantages of their approach in comparison to UDDI and Apache WSIF using a case study from the telecommunications domain.

In [17], 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.

6. CONCLUSION

According to the above analysis, we can see that SOA environment is defined in terms of a service. In the same sense, SOA is: A Service Oriented Architecture that arranges a business environment where human and technical efforts and/or capabilities are employed in order to guarantee the informational, practical, outgoing requirements and needs.

In summary, a service-oriented business environment in accordance with the principles of SOA will fail without a 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 I 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.

- 259 • Routing messages, in case of breaking or non-breaking versioning, targeted to old
260 versions to the newer versions.
- 261 • Deprecating then removing old services in an organized manner.

262 REFERENCES

263

264 [1] Christian Emig, Kim Langer, Karsten Krutz, Stefan Link, Christof Momm, Sebastian
265 Abeck. "The SOA's Layers". C&M Research Report, Karlsruhe, Juli 2006

266 [2] Seifedine Kadry. "An Implementation of ODBC Web Service. Journal of Theoretical and
267 Applied Information Technology. Vol 26. No. 1. 2011.

268 [3] <http://www.w3.org>

269 [4] Aleksander Dikanski, Sebastian Abeck. A View-based Approach for Service-Oriented
270 Security Architecture Specification. The Sixth International Conference on Internet and Web
271 Applications and Services (ICIW 2011), St. Maarten, Netherlands Antilles, März 2011.

272 [5] Kadry, Seifedine; Smaili, Khaled. A Solutions for Authentication of Web Service Users.
273 Information Technology Journal;2007, Vol. 6 Issue 7, p987.

274 [6] F Hazzaa, S Kadry. New System of E-Voting Using Fingerprint. International Journal of
275 Emerging Technology and Advanced Engineering 2, 355-363. 2012.

276 [7] David A Chappell, Tyler Jewell. Java Web Services. Publisher: O'Reilly Media Final
277 Release Date: March 2002. Pages: 278.

278 [8] <http://www.wikipedia.org>

279 [9] "Policy based governance for the enterprise," A WebLayers white paper, 2005.

280 [10] Barry, D. Web services and service-oriented architectures. 2004. Retrieved April 14,
281 2006, from <http://www.service-architecture.com>

282 [11] Ingo Pansa, Felix Palmen, Sebastian Abeck, Klaus Scheibenberger. A Domain-Driven
283 Approach for Designing Management Services. The Second International Conferences on
284 Advanced Service Computing (SERVICE COMPUTATION), Lissabon, Portugal, Oktober
285 2010.

286 [12] John Evdemon, "Principles of Service Design: Service
287 Versioning".<https://msdn.microsoft.com/en-us/library/ms954726.aspx>. August 2005.

288 [13] Hongqi Li; Zhuang Wu, "Research on Distributed Architecture Based on SOA,"
289 Communication Software and Networks, 2009. ICCSN '09. International Conference on ,
290 vol., no., pp.670,674, 27-28 Feb. 2009

291 [14] Mike P. Papazoglou. The Challenges of Service Evolution. 20th International
292 Conference, CAiSE 2008 Montpellier, France, June 16-20, 2008

293 [15] Bin Sarib, A.S.; Haifeng Shen, "SORC: Service-Oriented Distributed Revision Control for
294 collaborative web programming," Computer Supported Cooperative Work in Design
295 (CSCWD), Proceedings of the 2014 IEEE 18th International Conference. pp.638,643, 21-23
296 May 2014.

- 297 [16] Leitner, P.; Michlmayr, A.; Rosenberg, F.; Dustdar, S., "End-to-End Versioning Support
298 for Web Services," Services Computing, 2008. SCC '08. IEEE International Conference,
299 vol.1, pp.59,66, 7-11 July 2008
- 300 [17] G. Bechara, "Web services versioning," Apr. 2007. [Online]. Available: [http:](http://www.oracle.com/technology/pub/articles/web_services_versioning.html)
301 [//www.oracle.com/technology/pub/articles/web_services_versioning.html](http://www.oracle.com/technology/pub/articles/web_services_versioning.html)
302