RESTful Web Services Composition & Performance Evaluation with Different Databases

Neha Singhal
Dept. of ISE
Rajarajeswari College of Engineering
Bangalore,INDIA
neha.banasthali@yahoo.co.in

Usha Sakthivel
Dept. of CSE
Rajarajeswari College of Engineering
Bangalore, INDIA
sakthivelusha@gmail.com

Pethuru Raj
SRE Division
Reliance Jio Infocomm.ltd(RJIL),
SARGOD imperial, 23, residency Road
Bangalore, INDIA

***Abstract* - Service composition is a popular mechanism to orchestrate different and distributed services to produce composite services. Composed services are typically business-centric, and process-aware. There are a variety of use cases mandating such kinds of service compositions. With services emerging as the most appropriate building block and the unit of deployment, finding, composing, binding and leveraging services turns out to be an important job for software architects and developers. However, service composition is not an easy affair. There are several methods proposed by various researchers and scholars across the globe. Several parameters and considerations are being made in order to simplify and streamline service composition.**

Predominantly there are SOAP and RESTful services. There are markup languages to describe and define the distinct capabilities of participating services. In the recent past, a new architectural style (Microservices architecture (MSA)) is emerging and evolving fast. Microservices are lightweight, autonomous, self-defined, horizontally scalable, interoperable, and composable. In this paper, we have leveraged different databases for different services and evaluated the service performance individually and collectively. That is, multiple services are used and each service uses a database instance.

***Keywords*:** SOAP, RESTful, Composition of services

I. INTRODUCTION

In simple words, services are small in size and scope. Services are the most optimized and organized building block for producing all kinds of software applications. Services provide number of features:

- Platform independent
- Language independent
- Well defined
- Loosely coupled
- Ease of integration
- Self-contained
- Reusable

Services, when clubbed together, are bound to create new pioneering and promising applications. Producing newer applications start with the realization of heterogeneous services using different technologies, tools, programming languages, etc. SOA (Service Oriented Architecture) is the application architecture based on services. There are a few important ingredients for developing, deploying, delivering and maintaining service-oriented applications and systems. Service registry stores all the details required for discovering services. Service producers and developers develop various services and host them in web or application servers. The service metadata is stocked in the service registry. Then there are service consumers / users requesting various services to be compared and used. The service ecosystem is on the growth path. That means there are a bevy of powerful tools, engines, adapters, connectors, drivers, etc., in the service environment. Further on, there are service composition platforms and toolkits in order to accelerate the realization of connected, integrated and composed / choreographed services.

Popularly there are SOAP and RESTful services. REST means Representational State Transfer Protocol. This runs over the open and public data transmission protocol (HTTP). SOAP stands for simple object access protocol. SOAP, being the first and foremost method for service description, invocation, and orchestration, has gone through a variety of improvements in the form of several service standards. UDDI (universal description, discovery and integration) is one such service standard. Then we have WSDL, which stands for web service description language.

However RESTful services are easy to produce and deploy. Every element is being seen as a accessible and workable resource. RESTful services give the service representation for each of the resources in our everyday environments. Not only software applications, packages, and libraries, but also all kinds of other commonly found and used elements such as our handhelds, wearables, implantables, consumer electronics, equipment, machines, etc. can be visualized as resources and presented as RESTful services. There are a few common actions / methods that can be applied on the participating resources in order to be useful and usable. In this paper, we have specifically focused on RESTful services and how they can be composed together to form bigger and better services dynamically.

A. The Salient Features of RESTful Services

We have explained why RESTful services are becoming very

978-1-5386-2361-9/17/$31.00 ©2017 IEEE

popular across the industry. Every entity willing to participate in the mainstream computing has to be service-enabled. The proven RESTful method came as a silver bullet to speed up the process of service-enablement. Not only producing services from the ground up but also it is also equally important to generate smart and sophisticated services through composition (orchestration and choreography). Multiple services need to be involved in order to embark on the composition process.

Also in near future, application development happens via configuration, customization and composition. The business expectations are changing frequently and hence the IT divisions have to respond fast with new IT capabilities, which can be obtained through the proven composition toolkits and techniques. Further on, the service implementation time and complexity get reduced sharply with the use of composition tools and tips.

B. The Service Description Language

The service functionality, features and facilities are being expressed and exposed via the WSDL standard. This standard is the widely supported and common way to describe the specification and syntax of messages. As noted before, messages are the perfect and precise way for establishing a seamless linkage between participating RESTful services.

The service composition happens in two ways: static and dynamic. There can be sequential as well as parallel service compositions. The static composition primarily gets accomplished manually. That is, the services to be composed have to be identified and the composition takes place. This is both time-consuming and tough job.

Service environments are increasingly dynamic and hence services ought to be enabled accordingly through some kinds of knowledge such as policies and rules. That is, at runtime, services find the required services and collaborate with them to fulfill the professional as well as personal requirements. Automatically services find, choose and leverage other services to create composite services.

Precisely speaking, there are an arsenal of advantages for service orientation, management, security, science, and governance. Service composition plays a very vital role in shaping up service engineering. Automated service composition is the new normal with the availability of techniques. Different industry verticals focus on service composition to craft newer capabilities and capacities. Businesses are keen on utilizing the advancements in the service space in order to automate, accelerate and augment their service offerings.

II. PRELIMINARIES

The service composition challenges are many as articulated below.

- The number of services is continuously and consistently on the rise. Services are being curated, refined, and stocked in various service registries and repositories. Searching for a particular service is a tough task. For fulfilling certain tasks and goals, services need to be identified and subjected to composition to arrive at the required software solution.

- Secondly, services are dynamically created and uploaded into the service registries. That is, services are simply dynamic and hence have to have a mechanism in place in order to dynamically find and bind services to create newer software solutions. Runtime decisions are very important for the ensuing knowledge world.
- Thirdly, different service providers follow disparate conceptual models and hence there is a mandate for a kind of standardization in order to enable multiple service consumers to consume services. Thus there is a need for one uniform structure.

The two approaches in service composition are the centralized dataflow and the decentralized dataflow. For automated service composition, these approaches have both advantages and disadvantages. The prime limitation of the centralized dataflow is that all the participating services have to be routed through a composite service (service orchestrator). This can be a potential single point of failure. That is, each service message from the origination service to the target service has to pass through this service / message middleware. This passage results in higher response time. The performance and throughput issues crop up here. There are a few disadvantages being associated with the decentralized dataflow approach also. That is, each service directly shares data with its own web servers. The implications are the increase in the load at each node. This can result in copious delay in responding. The throughput (the number of operations per second) is also affected. However, the decentralized dataflow is highly efficient for accomplishing dynamic service composition. There is no tight coupling between the participating services. The elimination of dependencies brings forth a lot of benefits for service engineering. As there is no middleware involved in invoking services to team up with one another, the tight coupling between service clients and servers is avoided. In the proposed model, we have used the decentralized dataflow model approach, which results in high throughput, minimum response time and zero delay.

There are user interface (UI)/ graphical user interface (GUI)/ control line interface (CLI) / application programming interface (API)-based service composition. The other

compositions include functionality/application/process-based service integration. The primary requirement is functional integration so that the resulting composite services fulfill all the wanted service functionalities.

III. RELATED WORK

As articulated above, REST is an architectural style in which services are being viewed as resources. Each resource is being uniquely and universally identified through an identifier. URI (Universal Resource Identifier) is the mechanism through which RESTful services are being identified, accessed and assessed. The usage of HTTP methods helps in performing the standard PGPD operations (post, get, put, Delete). The basic REST design principle uses the HTTP protocol methods for the above four operations.

- *POST* - Create a resource
- *GET* - Retrieve a resource
- *PUT* - Update a resource
- *DELETE* - Delete a resource

The REST architectural style typically describes the following six conditions while leaving the implementation of the individual components to the developers

- Client-server
- Stateless
- Cached
- Uniform interface
- Layered system
- Code on demand

As explained above, WSDL is the language for describing RESTful service capabilities. This is a formal language that can be understood and acte upon by compute machines. There are other machine-processable languages (WADL, WSDL2.0 and SA-REST, RIDDL) for RESTful services. For integration and composition, we need such kinds of languages as a formalized mechanism. The existing service description languages do not properly and precisely capture the resource centric nature of RESTful services. Their main focus is on the description of input/output as being done by traditional service description languages. The current languages ignore the description of the resources and the transitions of these resources. The consequence is as follows. These languages work at the interface description level. Due to these reasons, these languages fail to facilitate automated composition of individual RESTful services.

IV. PROPOSED TECHNIQUE

A. Methodology

The proposed model methodology is described as below:

- The services are declared in registries.
- Service requester sends the request for service.
- Translator converts the request from external to system.
- The request arrives at composition module. It checks the matching Engine from WSDBs for requested service. Result is sent to evaluator if it finds the desired interface base service composition.
- Evaluator evaluates these selected web services in two steps. In first step, it evaluates the web services on the basis of interface-based search, whereas in second step it performs the evaluation on the basis of functionality-based rule. After evaluation, it sends selected services to composer. The purpose of composer is to compose these component web services.
- If Matching Engine does not find requested service composition from web services database then it starts searching from web.
- Multiple registries result send to evaluator to display the result. Matching Engine also has address of various services which want to execute. The purpose of aging is that it maintains the updated information about web services as the contents are refreshed each time when aging time expires.
- Evaluator evaluates these web services based on interface based and functionality based rules.
- Composer composes this evaluated result and sends result to Executor. Executor executes these services and through translator results are sent back to requester.

V. IMPLEMENTATION& EVALUATION

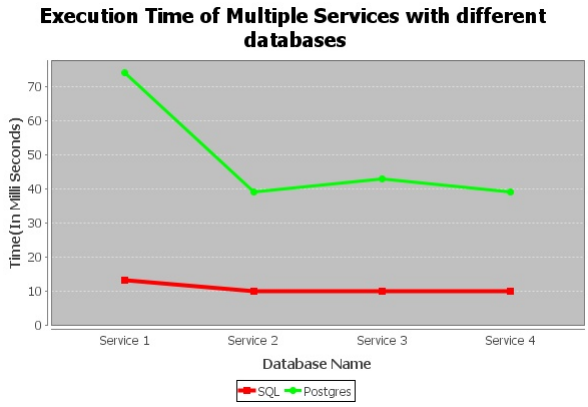
This paper describes the basics of creating an executable BPEL business process. The code for application is written in JAVA and deployed to the Apache Web Server.

In a distinctive scenario, the BPEL business process receives a user request. To fulfill it, the process calls the involved related services and then responds to the original user. Because the BPEL process shares information with other services, it depends on the WSDL descriptions of the Web services the invocation is handled by the composite service. To understand how business processes are described with BPEL, we will define a simplified business process for a hospital services website. The client invokes the query according to the hospital centre and give his/her name. We assume that hospital site provides a service through which we search doctors and the hospital services. Finally, the BPEL process gives the list of hospital centers to the client. We build a synchronous BPEL process. The new BPEL composite Web service uses a set of different port through which it provides functionality like any other services. We execute different services on different

databases and then the BPEL gives the performance data; with the help of these data we create the chart as shown in Fig 1.

To develop a BPEL process, go through the following steps:
1. Define and declare the related services.
2. Define the WSDL for the BPEL process.
3. Identify partner link types.
4. Define the BPEL process.
5. Build the new application.
6. Deploy the application.
7. Performance evaluation for execution time of multiple services with different databases.

Fig. 1. *Represents the Comparison of different databases*



VI. CONCLUSION & FUTURE WORK

In this paper, we have discussed about the main problems in fulfilling dynamic services composition. This paper has described our implementation of dynamic services composition algorithm. This work tries to solve the composition issues related to data distribution, reliability, availability and other quality of service (QoS) attributes. We have used two database instances for evaluating the performance of these RESTful services. The first database system is the MySQL DBMS and the other is the POSTGRES database management system. Finally, a graph is drawn for calculating the performance results of the services with the different databases.

REFERENCES

[1] Jonathan Lee, Senior Member, IEEE Computer Society, Shin-Jie Lee, and Ping-Feng Wang, "A Framework for Composing SOAP,Non-SOAP and Non-Web Services" IEEE TRANSACTIONS ON SERVICES COMPUTING, VOL. 8, NO. 2, MARCH/APRIL 2015

[2] FarzadKhodadadi,AmirVahidDastjerdi, RajkumarBuyya, "Simurgh: A Framework for Effective Discovery,Programming, and Integration of Services Exposed in loT" 2015 International Conference on Recent Advances in Internet of Things (RioT) Singapore, 7-9 April 2015

[3] Mahdi Bennara, Youssef Amghar, Michael Mrissa "ManagingWeb Resource Compositions" 2015 IEEE 24th International Conference on Enabling Technologies: Infrastructures for Collaborative Enterprises 978-1-4673-7692-1/15

[4] Guo Chen, Jiwei Huang, Bo Cheng, Junliang Chen," A Social Network based Approach for IoT Device Management and Service Composition" 2015 IEEE World Congress on Services

[5] Yang Xue, Chunhong Zhang and Yang Ji," RESTfulWeb Service Matching Based on WADL" 2015 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery

[6] Digvijaysinh M. Rathod, Dr. M S Dahiya, Dr. Saty en M. Parikh," Towards Composition of RESTful Web Services" 6th ICCCNT – 2015 July 13 - 15, 2015, Denton, U.S.A

[7] Konstantinos M. Giannoutakis, Dionysios D. Kehagias and DimitriosTzovaras," A three-level semantic categorization scheme of Web Services" 2015 IEEE 8th International Conference on Service-Oriented Computing and Applications

[8] R.P.Sumithra, R.Sarath," TOWARDS RESTFUL WEB SERVICE COMPOSITION FOR HEALTHCARE DOMAIN" 2015 International Conference on Control,Instrumentation, Communication and Computational Technologies (ICCICCT)

[9] G.Selvakumar, B.Jayakaviya," A Survey on RESTful web services composition" 2016 International Conference on Computer Communication and Informatics (ICCCI-2016),Jan. 07-09,Coimbatore,INDIA

[10] UrjitaThakar, AmitTiwari, SudarshanVarma "On Composition of SOAP Based and RESTful Services" 978-1-4673-8286-1/16 2016 IEEE DOI 10.1109/IACC.2016.99

[11] Shang-Pin Ma*, Peng-Zhong Chen, Yang-Sheng Ma, and Jheng-ShiunJiang"CARSB Portal: A Web-Based Software Tool to Composing Service Bricks and RESTful Services as Mobile Apps" 978-1-5090-3438-3/16 2016 IEEE DOI 10.1109/ICS.2016.118

[12] Shang-Pin Ma, Ci-Wei Lan, Ching-Ting Ho, and Jiun-Hau Ye," QoS-Aware Selection of Web APIs Based on _-Pareto Genetic Algorithm" 978-1-5090-3438-3/16 2016 IEEE DOI 10.1109/ICS.2016.121

[13] Youngmee Shin, Wanki Park, Ilwoo Lee," Design of Microgrid Web Services for Microgrid Applications" 978-1-5090-4749-9/17IEEE ICUFN 2017

[14] Elyas Ben HadjYahia;, Laurent R_eveill_ere, Y_erom-David Bromberg, Raphael Chevalier, and Alain Cadot," Medley: An Event-Driven Lightweight Platform For Service Composition"

[15] Martin Garriga , CristianMateos , AndresFlores , AlejandraCechich , Alejandro Zunino "RESTful service composition at a glance: A survey" Journal of Network and Computer Applications (2016)