SELECTION MECHANISM OF MICRO-SERVICES ORCHESTRATION VS. CHOREOGRAPHY

Neha Singhal[1], Usha Sakthivel[1], Pethuru Raj[2]

[1]Department of Information Science and Engineering, Rajarajeswari College of Engineering, Bangalore, INDIA

[2] Reliance Jio Infocomm. Ltd (RJIL), SARGOD imperial, 23, Residency Road Bangalore, INDIA

ABSTRACT

Web services is a special case of a service-oriented architecture (SOA), which is, basically, a representation of web application's functionality. Web service is more of a generalized concept that implies whole functionality as a whole but Microservice handles only the single specific task. MSA is emerging as an excellent architecture style enabling the division of large and complex applications into micro-scale yet many services, each runs in its own process, has its own APIs, and communicates with one another using lightweight mechanisms such as HTTP. Microservices are built around business capabilities, loosely coupled and highly cohesive, horizontally scalable, independently deployable, technology-agnostic, etc. On the other side for the business dynamic requirement these microservices need to be composed for the realization of enterprise-scale, and business-critical applications. Service composition is combining various services together to provide the solution for the user dynamic queries. There are two methods for the microservice composition i.e. orchestration and choreography. In this paper,a health case study is performed for the selection mechanism of orchestration method and choreography method in various situation.

KEYWORDS

MSA, Composition of services, SOA.

1. INTRODUCTION

Lately, microservices architecture is gaining a lot of mind and market shares. Monolithic and massive applications are being continuously dismantled to be a pool of easily manageable and composable microservices. Application development and maintenance (ADM) service providers know the perpetual difficulties of building and sustain legacy applications, which are closed, inflexible, and expensive. The low utilization and reuse are other drawbacks. Enabling them to the web, mobile and cloud-ready is best with a number of practical challenges. Modernizing and migrating legacy applications to embrace newer technologies and to run them in optimized IT environments consume a lot of time, talent and treasure. Software development takes the agile route to bring forth business value in the shortest possible time. Software delivery and deployment are getting equally speeded up through the DevOps concept, which is being facilitated through a host of powerful automation tools and techniques. Now the software solution design also has to be accelerated in a risk-free fashion. Here comes the microservices architecture style and pattern. Ontology's are gaining more popularity to search based on keywords. [19] For

the similarity purpose more various tools are available but protégé is gaining the more popularity among all[18].

Microservices are also innately facilitating horizontal scalability. Microservices are self-defined, autonomous and decoupled [16].The dependency-imposed constrictions are elegantly eliminated thereby faults are tolerated and the required isolation is being achieved. Microservices development teams can independently deliver on business requirements faster [1]. However, there are some fresh operational challenges being associated with microservices-centric applications. Microservices ought to be dynamically discovered. On finding the network location addresses, the control and data flows have to be precisely routed to the correct and functioning microservices. There has to be a controlled and secured access to microservices, which need to be minutely monitored, measured and managed in order to fulfil the designated business targets can be attained. All kinds of logging and operational data have to be consciously and consistently collected, cleansed and crunched in order to extricate usable and useful operational insights in time. Microservices are increasingly containerized and powerful DevOps tools (continuous building, integration, testing, delivery and deployment) are being used for business-empowerment.

The various characteristics of microservices are given below:-

Micro in size – Microservices is an implementation of MSA design pattern. It is recommended to keep your service small as much as you can. Basically, a service should perform only one business functionality; hence it will be smaller in size and easy to maintain the changes.

Focused to specific task – each microservice is designed to deliver only one specific business task. While designing a microservice, the architect should be Concerned about the central task of the service, this is its final product service. By definition, one microservice should be complete in nature and should be concerned to deliver only one business property. A fine grained service should focus to particular business logic.

Autonomous – each microservice should be an autonomous business module of the entire application. Hence, the application becomes less dependable, which helps to reduce the maintenance cost. Every microservice is independent and autonomous by nature.

Heterogeneous by nature – Microservice supports heterogeneity and technologies to communicate with each other in one, which helps the developers to use the selected technology at the appropriate place. By implementing a heterogeneous system, one can obtain highly secure, improved speed and a scalable system.

Resilience:-Resilience is a property of isolating a software module. Microservice follows high level of Resilience in building methodology; hence whenever one unit fails ,still it remains reliable and it does not impact the entire business functionality. Resilience is another property which implements highly scalable and less coupled system. Addition of new functionality is not complex.

Ease of deployment – as the entire application is sub-divided into small piece of units, every component should be full stack in nature. All of them can be deployed in any environment very easily with less time complexity unlike other monolithic applications of the same kind.

Microservices can communicate with each other. The communication between microservices is a stateless independent and self-contained by nature [16]. Hence, Microservices can communicate

effortlessly. In the Microservice architecture, the Data is isolated. Each Micro service has its separate data mart.

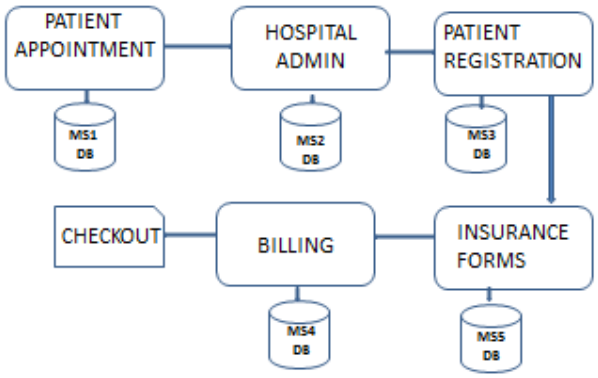


Fig.1. Microservices in healthcare domain

Delineating Containerization Paradigm in microservice:-

Containers emerge as the efficient runtime and resource for cloud applications (both cloud-enabled and native). Containers are comparatively lightweight and hence hundreds of containers can be made to run in a physical or virtual machine. There are other technical benefits such as horizontal scalability, portability, etc. Containers almost guarantee the performance of physical machines. Near-time scalability is seeing the reality with the faster maturity and stability of the enigmatic containerization paradigm. The tool ecosystem of containerization movement is growing rapidly and hence containers are being positioned as the perfect way forward to attain the originally envisaged benefits of cloudification.

Containers are being positioned as the most appropriate resource and runtime to host and execute scores of microservices and their instances. The container monitoring, measurement and management requirements are being speeded up with the availability of several open source as well as commercial-grade monitoring and data analytics solutions. The container networking and storage aspects are seeing a lot of tractions these days. Precisely speaking, there are a number of automated tools and viable approaches towards making containerization penetrative, participative and pervasive.

Why Containerization is pampered? - The *old way* to deploy applications was to install software applications on a bare metal server/physical machine (node/host) using the operating system (OS) package manager. This had led to the disadvantage of entangling the applications' executables, configuration, libraries, and other dependencies with each other and with the underlying host OS. With the fast maturity and stabilization of virtualization, the overwhelming practice is to build immutable virtual machine (VM) images to achieve predictable rollouts and rollbacks. But the main challenges include that VMs are heavyweight and non-portable.

The *new way* is to deploy containers, which implement OS-level virtualization rather than hardware virtualization. These containers are fully isolated from each other and also from the underlying host. The unique differentiations are that containers come with their own file systems and can't see other containers' processes. It is possible to bind the computational resource usage of each container. Containers are easier and faster to build than VMs. As containers are totally decoupled from the underlying infrastructure and from the host machine's

file system, they are extremely portable across local and remote servers. Also multiple OS distributions do not be a barrier for the container portability.

Containers are extremely lightweight. One application/process/service can be packed and hosted inside each container. This one-to-one application-to-container relationship brings up a bevy of benefits (business, technical and user). That is, immutable container images can be created at implementation time itself rather than at run time. This enables to generate different images for the different versions/editions of the same application. Bringing in technical and business changes into application logic can be easily accomplished and accelerated. Each application need not be composed with the rest of the application stack. Also application is not tied up with underlying infrastructure. Therefore, containers can run anywhere (development, testing, staging and production servers). Containers are transparent and hence their monitoring, measurement and management are easier to do. The key container benefits of containers are given below:-

- **Agile application creation and running** – Building container images through the techniques and tools provided by the open source Docker platform for containerization-enablement is faster. Not only development but also packaging, shipping and running containers are transparent, quicker and simpler.

- **Continuous integration, delivery and deployment** – The containerization concept has been hugely contributing for automating the DevOps tasks (continuous integration, delivery and deployment).

- **Separation of concerns between development and deployment** – As indicated above, it is possible to create container images at the build/release time itself. The deployment is totally decoupled from the development and hence applications can run on any system infrastructure without any hitch or hurdle. That is, containerization fulfils the longstanding goal of software portability.

- **Observability** – With the containerization paradigm, not only OS-level information and metrics, but also application-level information such as the performance/throughput, health condition, and other value-adding and decision-enabling details can be collected, cleansed and crunched to extricate actionable and timely insights.

- **An Optimal Runtime for Microservices –** Both cloud-native as well as enabled applications are predominantly microservices-centric. Containers are being positioned as the most optimal runtime for microservices. The convergence of containers and microservices is to bring a variety of benefits for cloud IT environments.

- **Resource isolation** – Due to the isolation brought in through containerization, application performance can be easily predicted.

- **Resource utilization** – Due to the lightweight nature of containers, accommodating many containers in a single machine is possible. Thus containerization leads to heavily dense environments. Further on, the resource utilization goes up significantly.

Containerization, without an iota of doubt, is being prescribed as the strategically sound tool for resolving most of the ills plaguing cloud environments.

2. OPPORTUNISTIC SERVICE COMPOSITION

Microservice composition is the method to combine various small services together to provide the solution of user complex business needs. Service composition means collaboration of services to build your software product [3]. Basically, it deals with high level software architecture paradigm where different modules of services will communicate for specific business goals.

Classification approaches in QoS-aware service composition

The service composition algorithms are classified in to three categories:

Non- heuristic:-Non heuristic algorithms are exact and accurate algorithms they provide optimal solution. The algorithms like Dynamic approach, Greedy approach, Divide and conquer approach, Bellman ford algorithm, Dijkastra algorithm falls under the optimal algorithm category.

Heuristic algorithms: - these algorithms are based on trial and error method. Heuristic algorithm provides fast results but not well for optimal solutions. Hill-climbing, Best first search, A* search algorithm, Pruning algorithm comes under this category.

Meta-heuristic algorithm: - these are broad range algorithms used for generalized solutions. Ex: - Genetic algorithm, Swarm optimization, PSO algorithm

2.1. Methods for service Composition

Microservice composition is the way to combine Microservice units to provide the solution for the complex requirements at a single glance. Microservice Compositions can be performed in two ways:-

- **Orchestration-based composition method.**
- **Choreography-based composition method.**

2.1.1 Orchestration Based Microservices Interaction

Orchestration is the most trusted and applicable way of handling interactions between different services in Service-Oriented Architecture (SOA). In orchestration, there is typically one central

controller that acts as the “orchestrator” of the overall service interactions. This follows a request/response message pattern. Only the central controller is responsible for all the interactions. The central controller the orchestrator is responsible for overall communication.

In the health care application, we have developed the microservice namely as given below:-

- Patient pre authentication [MS1].
- Eligibility and benefits process [MS2].
- Claims submission [MS3].
- Payment posting [MS4].
- Denial management [MS5].
- AR follows up [MS6].
- Reporting [MS7]
- Litigation[MS8]

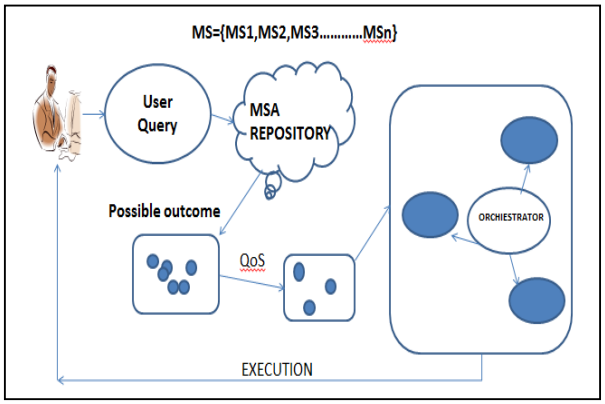


Fig.2. Microservice Composition With Orchestration Method

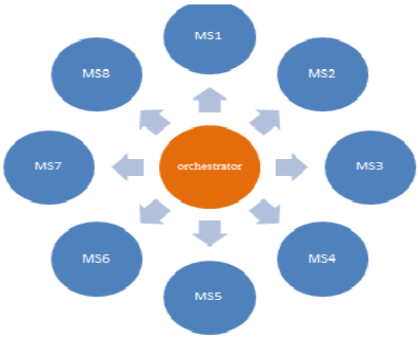


Fig: 3. Orchestration flow

In this approach of microservice orchestration, the orchestrator calls to one service and wait for the service response before calling to the next service. Once the called service response will come it will call to the other service with is required for the interaction. In this approach all the request will be send by the orchestrator and the next service call decision will be proceed by the orchestrator.

Benefits

Orchestration provides a effective way for coordinate the flow of the application when there is synchronous processing. For example, if Service A needs to complete successfully before Service B is invoked.

Trade-off

Dependency:-Coupling of the services together creating dependencies between the coupled services. If one service A is down, other service B and C will never be called. Service dependency is the concerned point in the centralized environment.

Single point failure: - orchestrator is a single point control and single coordinator. If it goes down, all processing stops and application fails.

Extra execution time:-Leverages synchronous processing that blocks requests. In this example, the total end-to-end processing time is the sum of time it takes for Service A + Service B + Service C to be called.

2.1.2 Choreography Based Microservice Interaction

In Microservice architecture, we want to avoid dependencies of one Microservice on other service. Microservice meaning each service should be able to represent on its own in independent manner. Reactive architecture patterns solve for some of the challenges of orchestration approach of composition.

Reactive architecture is considered as an event-driven architecture pattern applied to microservices. Instead of having a central orchestrator that controls the logic of what steps happen when that logic is built into each service ahead of time is referred as coordination or choreography. The services know what to react to and how, ahead of time, like a autonomous approach. Services use an event stream for asynchronous communication of events. Multiple services can consume the same events, do some processing, and then produce their own events back into the event stream, all at the same time.

The asynchronous nature of a reactive architecture removes the blocking or waiting time that happens with orchestration (request/response) type processing. Services can produce events and keep processing. Using an event stream for this enables communication between producer and consumers to be decoupled. The producer doesn't need to know if the consumer is up and running before they produce an event, or if the consumer received the event that was produced.

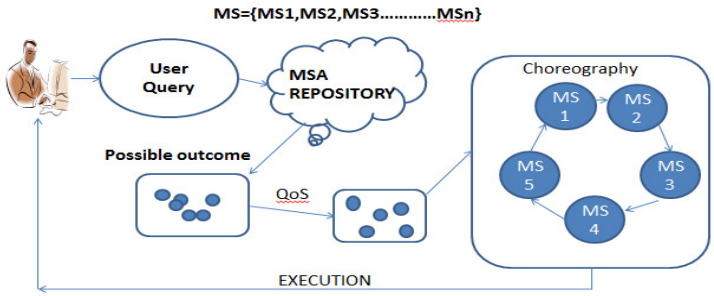


Fig:4.Microservice Composition with Choreography Method

Fig:5. Choregraphy for health care application

Also, in some cases, producers may want to direct commands to a specific service and receive acknowledgement that the consumer received it. Additionally, consumers/producers may want to consume/produce events from/to the event stream. This is a valid pattern and often you will find both approaches used together in a reactive architecture.

Benefits

- Choreography supports faster processing as services can be executed in a parallel fashion without depending on central controller service.
- Easier to add and update services as they can be plugged in or out of the event stream easily.
- Aligns well with an agile delivery model as teams can focus on particular services instead of the entire application
- Control is distributed, so there is no longer a single central controller i.e. orchestrator serving as a single point of success.
- Several patterns can be used with a reactive architecture to provide additional benefits. For example, Event Sourcing is when the Event pipeline contains all the events and enables event Re- execute. This way, if a service failure occurs while events were still being processed, then it came back and replays those events to get recovered. This enables each of these to be scaled independently.

Tradeoffs

- Async programming is often a significant mind shift for developers. At the particular time, the choreography can be performed in various ways.
- Complexity is again a concern point. Instead of having the centralized flow control in the orchestrator, the flow control is now divided and distributed across the individual services. Each service would have its own flow logic, and this logic would identify when and how it should react based on specific data in the pipeline event stream.

3. THE PROBLEM DESCRIPTION

Some researchers already performed analysis on how microservices choreography and orchestration techniques used for implementing micro service architecture. Here we are going to explain it in detail in the health care application.

To make healthcare application more effective and impressive, we need to perform micro service composition for the dynamic need and evaluate its processes for further improvements. Microservice process modelling opens up new challenging questions i.e. *which service mechanism is good for service collaboration.*
It's a big challenge to identify which composition approach is better among orchestration and choreography.

The purpose of this paper is to perform analysis on various considerable parameters like time consumption, power consumption, and memory consumption in orchestration and choreography process. The suitability of microservice orchestration and choreography is discussed in various scenarios of modelling.

4. IMPLEMENTATION & EVALUATION

This paper describes the basics of creating an executable BPEL business process for healthcare application. The application code is written in Java and deployed to the Apache Web Server.
In a healthcare scenario, the BPEL business process receives a user request. To fulfil it, the process calls the involved user request related services from the service registry and then responds to the requester client. The invocation is handled by the composite service. To explain BPEL, we defined a simplified business process for a healthcare application. The client invokes the query according to the healthcare centre and gives the suitable solution. We assume that healthcare application provides a service through which we search doctors and appointment booking related services. Finally, the BPEL process gives the list of requested services to the client. We build a synchronous BPEL process. The new BPEL composite micro service uses a set of different port through which it provides functionality like any other newly designed services. We execute service composition in two different ways namely orchestration and choreography.
The proposed model methodology is described as below:

- Service clients/consumers send requests to avail one or more services to the UDDI microservice repository.
- Setting up a micro service Registry like UDDI for microservices.
- Setting up a cloud Hub for stocking of the micro services participating in the service ecosystem.

- The selection criteria such as the typical QoS attribute (scalability, availability, performance/throughput, security, extensibility, compos ability, etc.) for selecting various micro services.
- The involvement of the Ontology technology for automatically selecting and using right and relevant micro services for business processes. The human intervention, interpretation and instruction are not needed for choosing the appropriate services to be composed.
- Orchestration or choreography for composing the chosen micro services in order to craft composite applications/workloads/processes.
- Composing multiple micro services leads to event-driven applications.
- The performance evaluation for the composite framework.

In this paper, various comparison analyses is performed for micro service choreography and microservice orchestration methods based on the following three parameters namely

- Time consumption
- Memory utilization.
- Power utilization

Configuration of machine:

- Windows 8.x
- Windows Server 2008 R2 SP1 (64-bit)
- RAM: 128 MB
- Disk space: 124 MB for JRE; 2 MB for Java Update
- Processor: Minimum Pentium 2 266 MHz processor
- Browsers: Internet Explorer 9 and above, Firefox

- Eclipse with Spring
- Apache Tomcat 7.0
- Postgres 9.0
- MYSQL 5.17

Time performance

Microservice composition is performed using service orchestration and the service choreography method and the time taken for execution in both the cases is observed for the various micro service executions.

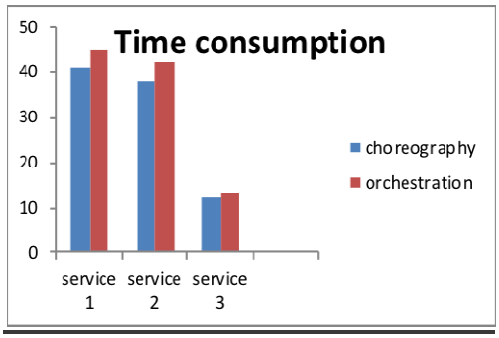


Fig.6.Time Based performance Evaluation

The analysis shows that the time consumption for the microservice choreography is less compare to the orchestration method.

Memory utilization:

Monitored the memory utilization for both the approaches i.e. microservice orchestration and choreography

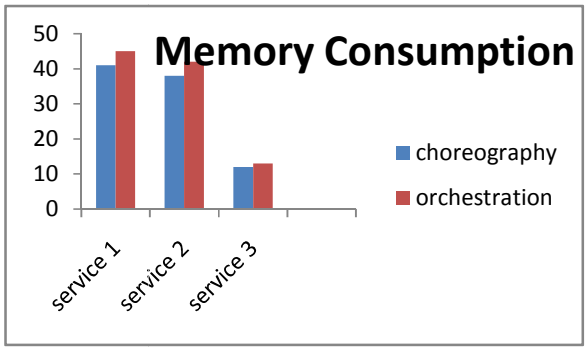


Fig.7.Memory consumption of different services

The analysis shows that the memory consumption for the microservice choreography is less compare to the orchestration method.

Power utilization:

Power consumption is estimated for the service composition methods.

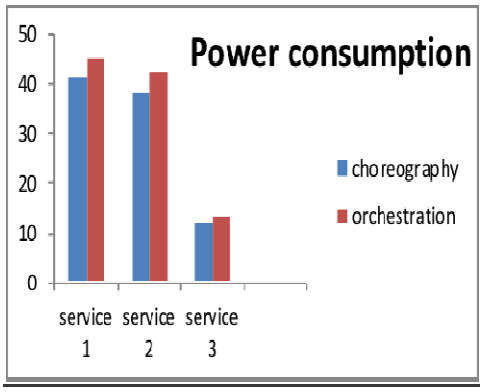


Fig.8.Power based consumption

The analysis shows that the power consumption for the microservice choreography is less compare to the orchestration method.

5. CONCLUSION & FUTURE WORK

In this paper, we described our implementation of dynamic services composition approaches i.e. using orchestration and choreography. This work tries to perform the analysis between both the

approaches orchestration and choreography. The analysis is performed based on time consumption, memory consumption, power consumption.

With the analysis based on time consumption, memory consumption, power consumption,
We were able to clearly identify that event choreography is much faster in performance as per the graphs, In comparison to orchestration.

However, event choreography becomes very complex to code and handle if there are multiple events triggered from each micro service. It is also evident that handling multiple actions for the triggers without a central orchestrator is tough as one developer or team working on a micro service may not be known to the other developer. This shows choreography based composition is a suggested approach when there is less number of micro services participating in the distributed business process, or the number of event triggers is not too many or when the trigger actions are not too complex. Orchestration is slow, but it is useful when the transaction scenarios are complex.

If application is not more complex and handling heterogeneous services is easy and maintainable than choreography is a suggested approach, But if application complexity is high and it's better to opt for orchestration as central coordinator in a single point of control and easy to maintain.
In future work, both choreography and orchestration can be considered together to perform more dynamism.

REFERENCES

[1] Damian Arellanes , Kung-Kiu Lau" D-XMAN: A Platform For Total Compositionality in Service-Oriented Architectures" 2017 IEEE 7th International Symposium on Cloud and Service Computing DOI 10.1109/SC2.2017.55

[2] Kleanthis Thramboulidis, Danai C. Vachtsevanou, Alexandros Solanos" Cyber-Physical Microservices An IoT-based Framework for Manufacturing Systems" 2018 IEEE .

[3] Damian Arellanes and Kung-Kiu Lau" Exogenous Connectors for Hierarchical Service Composition" 2017 IEEE 10th International Conference on Service-Oriented Computing and Applications" DOI 10.1109/SOCA.2017.25

[4] Chris peltz"web service orchistration and choreography" IEEE Computer socity,2003.

[5] Festim Halili,Eip Rufati , Ilia Ninka "Styles of Service Composition – Analysis and Comparison Methods " 2013 Fifth International Conference on Computational Intelligence, Communication Systems and Networks.

[6] Tanveer Ahmed, Abhishek Srivastava "Service Choreography: Present and Future" 2014 IEEE International Conference on Services Computing DOI 10.1109/SCC.2014.126.

[7] Sirine Rebai, Hatem Hadj Kacem, Mohamed Karˆaa , Saul E. Pomares and Ahmed Hadj Kacem1," A Service-Oriented Architecture (SOA) Framework for Choreography Verification" IEEE, ICIS 2015, June 28-July 1 2015, Las Vegas, USA 978-1-4799-8679-8

[8] Junio C. Lima Ricardo C. A. Rocha , Fabio M. Costa," An Approach for QoS-Aware Selection of Shared Services for Multiple Service Choreographies" 2016 IEEE Symposium on Service-Oriented System Engineering.

[9] Ján Terpák, Pavel Horovák, Matej Luká," Mathematical models creation using orchestration and choreography of web services" 2016 IEEE 978-1-4673-8606-7.

[10] Nacera Temgli,Abdelghani Chibani, Karim Djouani, and Mohamed Ahmed Nacer," A Distributed Agent-Based Approach for Optimal QoS Selection in Web of Object Choreography" IEEE SYSTEMS JOURNAL, VOL. 12, NO. 2, JUNE 2018, 1937-9234

[11] Lei Chen* and Cristofer Englund," Choreographing services for smart cities: smart traffic demonstration" 2017 IEEE 978-1-5090-5932-4.

[12] UrjitaThakar, AmitTiwari, SudarshanVarma "Choreography-based vs Orchestration-based Servic Composition in Opportunistic Networks" 2017 IEEE 13th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)

[13] 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

[14] 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

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

[16] 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"

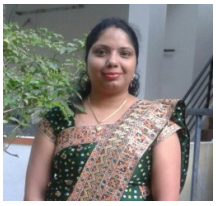
[17] Martin Garriga , CristianMateos , AndresFlores , AlejandraCechich , Alejandro Zunino "RESTful service composition at a glance: A survey" Journal of Network and Computer Applications (2016)

[18] Bhaskar Kapoor1 and Savita Sharma2" A Comparative Study Ontology Building Tools for Semantic Web Applications" International journal of Web & Semantic Technology (IJWesT) Vol.1, Num.3, July 2010

[19] H. H. Kian1 and M. Zahedi2" AN EFFICIENT APPROACH FOR KEYWORD SELECTION; IMPROVING ACCESSIBILITY OF WEB CONTENTS BY GENERAL SEARCH ENGINES" International Journal of Web & Semantic Technology (IJWesT) Vol.2, No.4, October 2011

[20] Q. Z. Sheng, X. Qiao, A. V. Vasilakos, C. Szabo, S. Bourne, and X. Xu, "Web services composition: A decade's overview," Information Sciences, vol. 280, pp. 218–238, Oct. 2014.

AUTHORS



Mrs. Neha Singhal is having 9.5 years of teaching experience, presently working as an Assistant Professor, Dept. of Information Science and Engineering since 2010. She obtained her M.Tech from banasthali university, Rajasthan. She is pursuing her Ph.D under VTU in the area of web services. Her teaching and research interests are in the field of web services.

S. Usha, is working as a professor and head , CSE, RRCE. Graduated from Manonmanium Sundaranar University, in Computer Science and Engineering during the year 1998. She obtained her Master degree in Computer Science and Engineering and PhD degree from sathyabama university in the area of Mobile Ad Hoc Networks in the year 2013. She has 54 publications in International and National conferences, 22 publication in national journal and international journals in the area of Mobile Ad hoc Networks and wireless security. Most of the publications are having impact factor citied in google scholar (h index and i10index), Microsoft etc.. Received fund from AICTE under NCP scheme and SERB(DST). Received best teacher award from Lions club in the year 2010&2012.Dveloped Centre of Excellence lab in IoT with industry collobration.Organized many conferences ,FDPs and Technical Talks. Associated with ISTE, CSI,IEEE,IAENG ,IDES and IACSIT.Reviewed papers in IJCs and CiiT jouranals. Acted as a TPC member in MIRA'14 IoTBDS '17 and IoTBDS'18 Portugal.. Chaired sessions in FCS'14, ICISC'13 & ICCCT'15., ICCCT'17and IoTBDS'18.

Pethuru Raj Chelliah, Finished the CSIR-sponsored PhD degree at Anna University, Chennai and continued with the UGC-sponsored postdoctoral research in the Department of Computer Science and Automation, Indian Institute of Science, Bangalore. Thereafter, I was granted a couple of international research fellowships (JSPS and JST) to work as a research scientist for 3.5 years in two leading Japanese universities. Published more than 30 research papers in peer-reviewed journals such as IEEE, ACM, Springer-Verlag, Inderscience, etc. Have authored 8 books thus far and focus on some of the emerging technologies such as IoT, Cognitive Analytics, Blockchain, Digital Twin, Docker Containerization, Data Science, Microservices Architecture, fog / edge computing, etc. Have contributed 30 book chapters thus far for various technology books edited by highly acclaimed and accomplished professors and professionals.