Global Scientific JOURNALS

GSJ: Volume 9, Issue 8, August 2021, Online: ISSN 2320-9186
www.globalscientificjournal.com

A REVIEW ON SOFTWARE ARCHITECTURAL PATTERNS

Ebenezer, Olukunle, Oyebode
Ireti, Mojisola Oyerinde

Author Details (optional)
Iretioluwa, Mojisola, Oyerinde is currently pursuing masters degree program in electric power engineering in University, Nigeria, 08032481502 E-mail: eo.oyebode@acu.edu.ng
Co-Author name is currently pursuing Ph.D degree program in Computer Science in LAUTECH University, Nigeria, 08033701102. E-mail: iretihope12@gmail.com
(This information is optional; change it according to your need.)

KeyWords
Architecture, Software, View, Style

ABSTRACT

Architectural patterns are used to address important aspects of software architecture and complement existing techniques. Architectural pattern offers a set of predefined subsystems with their responsibilities and syntax for organizing the relationships between them. There are many architectural style that can serve as basis for software development. This study examined the architectural style, their applications, strength and weaknesses.

MAIN PAPER STARTS HERE...

Software is becoming more and more pervasive and has come to stay as part of solution from technological background to everyday life because of its usefulness in solving problems that readily come with ease. Technologies are everywhere in society today and the number of online transactions and services via websites and mobile applications are growing at alarming rate[8]. Software can be used in nearly all fields of human endeavour including complex systems. In domains like cloud, big data, expert systems, internet of things (IoT), cyber-physical systems (CPS), software is the core element. Due to diverse use of software, the techniques for design of software are many. Various design patterns meant for software are part of software architecture. Software is a product of software architecture and software pattern(s).

In this study, the concept of software architecture and architectural patterns have been reviewed to draw distinction to the various architectural patterns including their areas of application, the strength of each technique and the weakness.

2 Software Architecture

A software architecture is a description of subsystems and components of a software system and the relationships between them. Subsystems and components are typically specified in different views to show the relevant functional and non-functional properties of a software system. The software system is an artifact. It is the result of the software design activity. A component is an encapsulated part of a software system. A component has an interface. Components serve as the building blocks for the structure of a system. At programming-language level, components may be represented as modules, classes, objects or as a set of related functions. A subsystem is a set of collaborating components performing a given task. A subsystem is considered a separate entity within a software architecture. It performs its designated task by interacting with other subsystems and components.

The architecture of a system describes its major components, its relationships (structures) and how it interacts with other subsystems. Software architecture provides a blueprint for a software development. It can give an understanding of key elements that should be put together to achieve development of a software. Software architecture gives a structured solution to meet all the technical operational requirements while optimizing the common quality [4].

Software architecture gives the means and techniques including the tools, methods and ideas to create fundamental system structure. [5] stated that software architecture presents the high-level structure for software systems with the views of the design deci-

sions and hides behind in form of abstract, the complexity of the low level details. Software architecture is a step in software development life cycle which defines how to solve the problem of the customer such that solution contains all the functional and non-functional requirements of the customer [4]. Software architecture can be used to increase the quality of the software, help the developer work with the requirements of the system and extend the lifetime and overall quality of the software[11]. The software architecture refers to the descriptions of a system that consists of basic structural elements that must interrelate to arrive at the expected structure.

Software architecture may be used to increase the quality of the software, help the developer work with the requirements of the system and extend the lifetime and overall quality of the software applications. Architectural requirements, architectural design, architectural description and architectural evaluation are examples of techniques and activities a software architect uses for the architectural design process.

The software engineering addresses both the functional and non-functional requirements of any system that is meant to be developed. The software architecture refers to the non-functional requirements while software design entails the functional requirements. A design pattern provides a scheme for refining the subsystems or components of a software system or the relationships between them. It describes a commonly recurring structure of communicating components that solves a general design problem within a particular context.

3. Architectural Patterns

An architectural Pattern creates a means of representing, sharing and reusing knowledge of designs meant for achieving standard software systems. An architectural pattern presents a fundamental structural organization schema for software systems. It is a stylized description of a good design practice which have been implemented and found useful in different environments.

An architectural pattern is a proven structural organisation schema for software systems. A pattern is a description of a set of predefined subsystems and their responsibilities (Onarcan, 2018). Software Architectural Styles can be described as principles which shape an application. It can be likened to an abstract framework of a system in the area of its organization. Architectural pattern offers a set of predefined subsystems with their responsibilities and syntax for organizing the relationships between them. Architectural pattern is of great importance in developing software. Architectural patterns offer a useful approach for developing software with defined properties. Patterns cover a wide range of scale and abstraction. Atimes, it can be interwoven with each other as some patterns can be used to refine another pattern or other larger patterns so as to solve more complex problems [6]. Architectural patterns are used to address important aspects of software architecture and complement existing techniques. It offers a mental toolbox that could be used to construct software that meets both the functional and nonfunctional requirements of an application. Pattern has been applied in diverse software domains such as business domain, the automation domain, telecommunication domain etc [8].

Architectural patterns have been used to express fundamental structural organization schemas for software systems. It offers a set of predefined subsystems, specify responsibilities for the subsystems and include rules and guidelines for organizing the relationships between them. Architectural patterns help to specify the fundamental structure of an application (Rimawi and Zein, 2018). Every development activity is controlled by the structure in the pattern. Some of the Architectural patterns in use for software development include:

(a) Layered/Tiered architecture: The Layered architecture is one of the simplest approach for dividing complicated software to simpler form for easy development. Each subsystem can be handled as layer and a layer above can consume the service of the layer below it without necessarily informing the layer below it. [15] stated that Layered architecture allows the grouping of the responsibilities of the software into several loosely coupled layers. Layered architecture is suitable for categories of applications that require strict standards of testability. Layered architecture is commonly used for enterprise applications that need developed quickly as it offers well-organized application. Each layer can communicate with each other via well-defined interfaces. The layered architecture can be used in standalone system or distributed systems over several computers [7]. It can be applied in machine architecture etc Layered architecture usually has three important layers which are:

(i) Presentation layer: This layer is responsible for user interface functionalities. It allows users to interact with the system. It has capability to accept user requests and render appropriate interface for the user to access depending on the application type.

(ii) Business layer: This layer handles the logic involve in carrying out operation or render service. It can also be called domain logic. It is responsible for coordinating the logic of the application or service.

(iii) Infrastructure layer: This layer incorporates data services, networking services and other services required for delivery of service or performance of application. A pattern that is usually engaged why following the layered architecture is the MVC.

Model-view controller pattern: The model view controller has three separate parts which are the model, the view and the controller. The pattern combines the three parts together for development. The model deals with the development of the main application. It defines the business logic of the application usually made-up of classes that define the domain of interest. The view deals with the presentation of data in the model in terms of design templates in form of look and feel. The controller deals with integrating the model part with the various views depending on the state of the application. The controller manages the relationship between a View and a Model. It responds to user requests, interacts with the Model and decides which View should be generated and displayed [14]. It gives room for richness of designs through benefits of the

look and feel, making multiple views from the same source etc.The weakness of MVC lies in the area of using MVC with modern user interface tools due to inevitability of change to view and controller when porting.

Examples of areas where layered approach has been used include open system interconnection (OSI) and transmission control protocol/internet protocol (TCP/IP), e-commerce web applications and desktop applications.
The strength of layered architectural patterns are:

(i) It gives room for reuseability
(ii) The dependencies for each layer can be kept local.
(iii) It supports standardization that allows software developed with the technique to be used across many platforms.

The weaknesses of layered architecture are:
The complexity and cost of adding more layers. The use of layered architecture can turn out to reduce efficiency in terms of delivery of service as it may emanate unnecessary work. It is efficiency depends on using the correct granularity of layers, using too many layers can result to unnecessary complexity and overhead.

(b) Event-driven architecture (EDA): EDA is an asynchronous message-driven communication model meant to disseminate information for an enterprise [2]. It allows business activities to be viewed as series of events. In a typical application/service, several events can be identified and placed in various categories. EDA pattern can tolerate event notifications that can lead to information dissemination and reactive business process execution. It can make use of asynchronous communication to propagate information to all services and applications. It has a core of single-purpose event processing components that can listen to events and process them asynchronously. Event driven pattern can use two types of topologies.

(i) Event mediator topology pattern: This uses central mediator to organize multiple steps that are part of an event. The four main components of the mediator topology are:
  - Event queue: This receives events from the clients and forward it to the event mediator.
  - Event mediator: This receives the event and orchestrates the events. The event mediator sends additional asynchronous events to all the event channels.
  - Event channels: This is used by the mediator to transport processing events to the event processor.
  - Event processor: The event processors can receive event from the event channel and process it by applying business logic to process the event.

(ii) Event broker topology pattern: In a situation where events identified in application is relatively simple, the event broker topology can be used. The structure has broker and event processor. The event broker is responsible for processing and publishing each event. The event broker may be implemented in centralized or federated form. There are some variants of event driven pattern which include: Hub and spoken pattern, broadcast pattern, pooling pattern etc.
Event-Bus pattern deals with events. The event sources will publish messages to particular channels on an event bus. Event listeners subscribe to particular channels. Listeners are notified of messages that are published to a channel to which they have subscribed. Generation and notification of messages is asynchronous: an event source simply generates a message and may then go on doing something else; it does not wait until all event listeners have received the message. Event driven pattern uses distributed asynchronous technique to improve on the challenge of distributed data. It is a scalable and flexible architecture that allows each micro service to publish event such as ordering of service or related modifications [9]. It allows interactions among micro service such that a microservice can subscribe to another micro service(s) that is of interest. Event represents a change in state.

Event-driven architecture is one of the techniques for implementing modern day cloud applications. It can also be used in android applications and notification messages. The use of event bus pattern has the challenge of cross platform support. Once implemented, it is usually permanent and without easy upgrade, it can become redundant.

(c) Microservice Architecture: Microservice pattern is a new architectural style that can be used for building software systems by composing lightweight services that perform very cohesive business functions [13]. A microservice refers to a fine-grained software unit that can be created, initialized, duplicated and destroyed independently from other microservices of the same system. Microservices has been viewed as the most powerful architectural pattern useful for designing, developing and deploying mission critical applications. They are usually loosely coupled. It gives support for multiple programming languages as can be obtained in polyglot model. Microservices operates in interoperable, technology-agnostic and are composable to produce process-centric applications [3]. Microservices patterns are supported by docker-enabled containerization, API gateways, integration platforms that made successful transformation of monolithic applications for effective service delivery [16]. Microservices architecture has been identified as the way forward to realize the dream of digital enterprises. Microservice architecture made it possible to visualize service as combination of interdependent microservices. It is very useful in building high scalable and flexible large scale and distributed software systems.

The microservices architecture can accommodate several other patterns in designing flexible and scalable applications. Such patterns include: Orchestration and Coordination Architectural Patterns**,** Aggregator pattern, API gateway design pattern, chain of responsibili-

ty pattern, branch pattern, and asynchronous messaging design pattern.

Orchestration and Coordination Architectural Patterns:

(i) The API-Gateway Pattern: API Gateway is the entry point in a system that is responsible for routing requests to appropriate microservices and gathering results from different services. It offers a tailored API to client for routing requests, transform protocols, implement shared logic such as authentification. Other functions include monitoring, static response handling. The API gateway pattern can be used for achieving load balancing.

(ii) Service Discovery Patterns: In microservives pattern, need may arise to allow multiple instances of the same microservice to run in virtual machines and containers. In such situation, the client must be able to recognize and efficiently communicate with the appropriate microservice. Service discovery patterns dynamically supports the resolution of DNS address into IP addresses. Clients can query the service registry, select available instance and make direct request.

(iii) Server-Side Discovery Pattern: When clients are allowed to make requests via a load balancer that can query registry and forward the request to available instance. This pattern permits clients and microservices to exchange communication directly.

(iv) Hybrid Patterns: The hybrid pattern is a combination of the service registry and API gateway where the API gateway is completely replaced with message bus. The clients engage in direct communication with the message bus which routes requests to the requested microservices.

Asynchronous Messaging Design Pattern.

The pipes and filter patterns:

Pipe and filter is a useful pattern for message patterns. Pipe and filter allows components (filters) where data can be subjected to modification with certain objectives and communications (pipes) can be used to allow the flow of data between components. The pipe and filter pattern breaks complex tasks into multiple subtasks such that each of them can be implemented in a separate independent filter. Filters can communicate with each other via data transfer using pipes. Each filter has the capability to operate concurrently and communicate with other filters [10]. When a single event can trigger multiple actions, pipe and filter can be used. It can be applied for processing complex messages. The filter represents a subtask that can be handled. Each filter has interface that possesses inbound pipe with capability to receive, process and publish result to outbound pipe. All the pipes are connected to the same external interface. The connection outlet can be referred to as port. Each filter can be identified as having one input port and one output port. The pipe and filter pattern can be used to enrich other patterns to achieve some useful purposes such as message router pattern. The communication cost must be regulated so that the communication cost will not out-weigh performance. Determining the number of filters that can maximize the processing capability of the hardware is also a challenge (Bi *et al.,* 2018).

The use of Pipes and Filters support rapid prototyping of pipeline.

It is efficient with parallel processing

It supports reusability of filter components.

The use of intermediate file may not be necessary

The use of pipe and filter has demerits in terms of the cost of obtaining the state information which is high and Its use is associated with high data cost.

Microservices can be deployed across heterogeneous execution platforms via network. Each microservice can implement some part or a single function of the business logic of applications [1]. It can easily allow the use of different databases technologies and independent testing of the databases. It can use hypertext transport protocol (HTTP) resource application programming interface (API) or indirectly by means of message brokers. It is easily deployable on virtual machine or light weight containers. Although, microservice architecture is useful in building complex systems, its weakness is in the aspect of security. It has many entry points that make security to become more challenging with the use of microservice architecture [12].

Conclusion

The use of architectural pattern has significant role to play in the design of software. Pattern can also be combined together in other to realize software in some cases. In this study, attention has been given to software architecture and some important samples under each category. Such study can be used to gain insight to know the more about architectural style as it adds to the body of knowledge and influence choice of particular pattern at design time.

Acknowledgment

The authors wish to thank A, B, C. This work was supported in part by a grant from XYZ.

References

[1] M. Fazio, A. Celesti, R. Ranjan, C. Liu, L. Chen, M. Villari, Open Issues in Scheduling Microservices in the Cloud. IEEE Cloud Comput. 3(5):81-88, 2016.
[2] E. B. H. Yahia, L. Reveillere, Y. D. Bromberg, R. Chevalier, A. Cadot, Medley: An Event-Driven Lightweight Platform for Service Composition. International Conference on Web Engineering pp. 3-20, 2016.
[3] P. D. Francesco, P. Lago, and I. Malavolta, Architecting with microservices: A systematic mapping study. The Journal of Systems and Software 150:77-97, 2019.
[4] A. Sharma, M. Kumarb and S. A. Agarwal. Complete Survey on Software Architectural Styles and Patterns. 4th International Conference on Eco-friendly Computing and Communication Systems 16 – 28, 2015.
[5] A. Hassan and M. Oussalah, Evolution Styles: Multi-View/Multi-Level Model for Software Architecture Evolution. *Journal of Software*, 13(3):146-154, 2018.
[6] L. Yu, Y. Li and S. Ramaswamy, Design Patterns and Design Quality: Theoretical Analysis, Empirical Study, and User Experience. *International Journal of Secure Software Engineering*, 8(2):53-81, 2017.
[7] P. Liang, B. Tingting, and A. Tang, "Architecture Patterns, Quality Attributes, and Design Contexts: How Developers Design with Them?". 25th Asia-Pacific Software Engineering Conference (APSEC) 49-58, 2018.
[8] M. Maier and R. Harr, "Dark Design Patterns: An End-user Perspective". *Human Technology*, 16(2): 170-199, 2020.
[9] S. Zhelev, and A. Rozeva, "Using Microservices and Event Driven Architecture for Big Data Stream Processing". Proceedings of the 45th International Conference on Application of Mathematics in Engineering and Economics (AMEE'19), 090010-1 - 090010-8, 2019.
[10] Y. Zhai and L. Zhang, "Formal Description of Pipes-filters Architecture Style". 3rd International Conference on Materials Engineering, Manufacturing Technology and Control (ICMEMTC 2016) 1216-1219, 2016.
[11] T. Bi, P. Liang and A. Tang, "Architecture patterns, quality attributes, and design contexts: How developers design with them". Proceedings of the 25th Asia-Pacific Software Engineering Conference (APSEC), pp. 49–58, 2018.
[12] H. Song, P. H. Nguyen, F. Chauvel, J. Glattetre, and T. Schjerpen. "Customizing Multi-Tenant SaaS by Microservices: A Reference Architecture". IEEE International Conference on Web Services (ICWS). 446–448, 2019.
[13] A. Osman, P. Bruckner, H. Salah, F., H., P. Fitzek, T. Strufe and M. Fischer, "Sandnet: Towards high quality of deception in container-based microservice architectures". IEEE International Conference on Communications (ICC) 1–7, 2019.
[14] S. Singh, "MVC Framework: A Modern Web Application Development Approach and Working". International Research Journal of Engineering and Technology (IRJET) 07 (1): 51-55, 2020.
[15] H. Washizaki, S. Ogata, and A. Hazeyama, "Landscape of Architecture and Design Patterns for IoT Systems". IEEE Internet of Things Journal 7(10):10091- 10101, 2020.
[16] D. Jaramillo, D. V. Nguyen and R. Smart, "Leveraging microservices architecture by using Docker technology". *SoutheastCon 2016*, pp. 1-5, 2016. doi: 10.1109/SECON.2016.7506647.