A Systematic analysis of Software Patterns and Quality Attributes in Software Architecture

Shivang Konde[a], Subham Anand[a], Priyansh Deshmukh[a], Amit Mehta[b], Shyamnand kumar[b]

[a]*Software System, BITS Pilani, Pilani, 333031, Rajasthan, India*
[b]*Computer Science, BITS Pilani, Pilani, 333031, Rajasthan, India*

Abstract

Software architecture patterns are universal and reusable solutions to commonly occurring problems in software architecture. Each architecture pattern describes the high-level structure and behavior of software systems and addresses a particular recurring problem within a given context in software architecture design. Architectural patterns are aimed to satisfy several functional and quality attribute requirements. Selecting appropriate patterns is challenging for software architects as the knowledge about patterns is scattered. This leads to inefficient design efforts and potentially poor architectural choices.So in this paper we conducted a systematic literature review of several software architecture and found some prominent software Patterns like Client-Server, Layers, Channels and Channels, Service-Oriented Architecture(SOA), and Model-View-Controller (MVC) and Quality Attributes like Reusability,Flexibility,Performance efficiency, Scalability and Maintainability.With the knowledge available, architects can more rapidly select and eliminate combinations of patterns to design solutions.

1. Introduction

1.1. Background

An architectural pattern within a given context is a reusable resolution to a commonly occurring problem in software architecture. They are similar to software design patterns but have a broader scope, encompassing the entire system architecture. In essence, architectural patterns provide generic

solutions to address issues related to quality attributes, and their impacts on these attributes are a key consideration in software architecture decision-making. A decision-making process has to be followed to select appropriate architectural patterns. This process may contain the following steps:

1. identifying a design problem.
2. looking for the exact features that can accommodate a solution to this problem.
3. Finding a suitable pattern among several patterns

employing tactics to implement said pattern within the context of the design issue. As the information on patterns is distributed among a vast literature, it becomes very hard to select a suitable pattern unless the information on the patterns is collected in an organized manner and is easy to navigate through. A systematic decision-making process can help software architects select patterns according to their requirements.

1.2. Motivation

The motivation for doing research analysis of architectural patterns is to collect the various information on patterns distributed among vast literature and provide a systematic literature analysis of architectural patterns and their impact on quality attributes. That would help architects select suitable Patterns for their respective software architectures and apply suitable tactics to improve the quality attributes. These patterns are reusable. So, conducting a thorough analysis and collecting information about various software patterns saves time for architects if a similar framework is used in designing a given software system. So selecting a suitable model is important for architects to save their time and it also provides a suitable framework for designing and implementing software systems. This motivates us to collect the scattered knowledge of architectural patterns and provides a systematic analysis of several architectural patterns.

1.3. Objective

In this paper we have conducted a systematic literature review on top 5 patterns of total 29 patterns and analysed them based on selected 5 quality attributes among 40 quality attributes mentioned in the paper.[1] Software Patterns described in this paper are:

1. Client-Server

2. Pipe and Filter
3. SOA
4. Model-View-Controller (MVC)
5. Micro-Services

Quality Attributes:

1. Reusability
2. Flexibility
3. Performance and Efficiency
4. Reliability
5. Maintainability

2. Software Patterns

In this section, we are analyzing different patterns.We studied the top 5 patterns like Client-server, Pipes and Filters, Service Oriented Architecture(SOA), Model-View-Controller (MVC), Micro Services, and then ordered them in descending order of the number of primary studies conducted for each pattern. [1]

2.1. CLIENT-SERVER

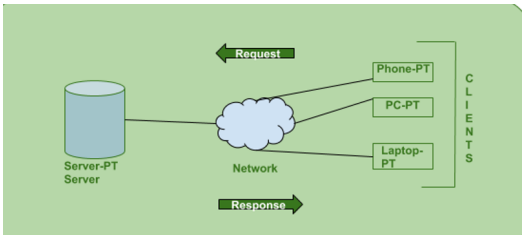

Figure 1: Client and Server Architecture.

In Client-Server there is a client and server in which the client requests the server and the server responds to the client's request. In this client-server pattern client always initiates the communication by requesting service from the server and the Server responds to the request made by the client.This

model is a Centralized architecture, that has both its benefits and demerits. The benefit is that since we have a single server, it becomes easier to manage data, and also it provides security by incorporating authorization or authentication to identify legitimate users and demerit is that on multiple client requests server gets slowed down and may even become a bottleneck. [2]

Quality Attributes satisfied by Client-Server Architectural Pattern are: Security: Client-Server provides more security as it provides authentication to check for legitimate users.

1. Scalability: The Client-Server pattern is neutral towards scalability because in this pattern we can add more clients and servers but up to a limit, if more numbers of clients are added then the server can become the bottleneck.
2. Re-usability: It also promotes re-usability.
3. Modularity: In this model, we have two independent modules client and server.
4. Interoperability: Communication Mechanism used in this model is based on HTTP, TCP/IP, and RESTful API. [3]

Strengths:

1. It is centralized, so it's far easier to hold, update, and secure the system.
2. It has high performance and occasional latency because the server can deal with many requests from many customers simultaneously and effectively.

Liabilities:

1. It has limited scalability because it relies upon the potential and availability of the server.
2. It relies on the network connection between the client and the server. If the community is slow or disrupted, the gadget may additionally enjoy delays or errors.

Uses: Client Server Model is used in File Transfer and Mail Transfer, It is also used in transferring images and files through HTTP.[4]

2.2. PIPES AND FILTERS

The Pipe-and-Filter (P and F) architectural style is a well-known and it is frequently used design pattern in software architecture. It consists breaking down a system into a series of processing elements also known as filters

Figure 2: Pipes and Filter.

that are connected by channels known as pipes through which data flows. Filters are generally answerable for processing the information, and pipes makes it easy to switch the records among filters. This architectural fashion promotes modularity, reusability, and parallelization, making it more appropriate for programs requiring high throughput and requiring low reminiscence footprint. The P and F fashion has been widely used in one-of-a-kind domains, together with dynamic evaluation, excessive-extent facts processing, and parallelization on heterogeneous platforms. The report discusses the Pipe-and-Filter (P and F) architectural fashion and introduces the TeeTime framework as a option to the restrictions of present P and F frameworks. It highlights the demanding situations in modeling and executing normal P and F architectures, the dearth of help for multiple input and output streams consistent with clear out, and the need for efficient parallel execution. TeeTime is shown as a framework that could model and execute any P and F architectures, allow parallel execution of filters, and offer extensibility for experimenting with the P and F style.

The paper emphasizes the importance of the P and F fashion in enterprise and research, especially in optimizing the execution on heterogeneous platforms. It also addresses the limitations of existing frameworks, together with their awareness on unique use instances and forget of type protection whilst interconnecting filters. TeeTime is placed because the first framework capable of modeling and executing arbitrary P and F architectures, permitting parallel execution of filters and supplying extensibility for experimenting with the P and F style.

The file searches the key functions of TeeTime, including its support for all P and F variations, parallel execution model with a high stage of abstraction to test with the P and F style. It also describes the automatic pipe choice, efficient multi-port composite levels of TeeTime. The paper consists of an example utility of TeeTime in Java and provides code listings to demonstrate the implementation of configurations and tiers the use of TeeTime.

In end, the document highlights the significance of the TeeTime framework in addressing the restrictions of current P and F frameworks and allowing the modeling and execution of arbitrary P and F architectures. It emphasizes the capability of TeeTime in advancing the development of P and F architectures and its applicability in numerous domains, offering a complete review of its key features and destiny research directions.[5]

2.3. SOA

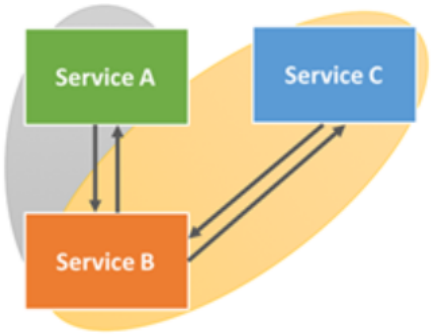

Figure 3: Software Oriented Architecture.

Service Oriented Architecture (SOA) is a type of architectural pattern or design, that provides services to different applications in distributed systems. These systems follow a standard-based, technology-independent computing paradigm for distributed systems.[6]

In SOA service provided has three important properties: The first one is that a particular service is independent of others, which means that it has its own data and operation it does not depend on other services. The second important property is that Services in SOA are platform-independent and the third one is that SOA assumes that services are dynamically located, invoked, and combined.

SOA provides flexibility and scalability So that we can build and incorporate many Software Systems by promoting the reuse and interoperability of services. SOA permits the corporation to create an agile and adaptable machine that could respond to distinct business needs and technological advancements.

In terms of Quality Attributes provided with the aid of SOA, It offers some of the fine attributes like: 1. Availability: The machine always responds to the request message from customers which represents that SOA-based total structures are usually available, 2. Reliability: It is easy for SOA-based systems to test and debug as they are smaller in size and it also provides Security. Since each service or application is independent of each other then change in one service does not impact the other. So in this way, SOA provides Loose Coupling [7], and as it is layer-based architecture it also provides parallelisation.

There are several present-day applications of SOA architecture like in gaming, the Use of GPS, and also in a military organisation.

2.4. Model View Controller

MVC is a design pattern that helps organize and structure software applications. Its basic features are modularity, scalability, and maintainability. It helps in managing and modifying different components of applications.

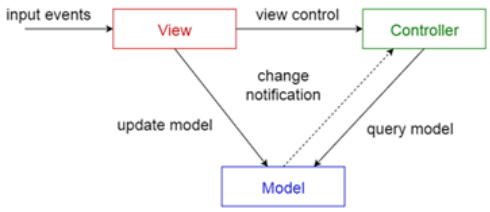

Figure 4: MVC Architecture

Key components of the Model View Controller are :

1. Models: These are like the data keepers. They hold information that users work with. It could be simple stuff like numbers or more complex things like objects. Models can be either: 1)View Models: These transfer data between views and controllers. 2)Domain Models: These contain data specific to a business domain as well as the rules for manipulating that data. [8] They also contain the business logic of the application which defines the rules and operations to manipulate data.
2. Views: Represents the user interface and displays the data from the model. [8] It can be considered a visual layer that converts the data into a user-friendly format. There can be multiple views associated with a single Model View Controller Architecture. Views must be dynamic and have an associated controller component.

3. Controllers: Controllers are responsible for processing requests, performing various operations on the model, and selecting the views to be rendered to the user. [8]

Users interact mainly through the Views and their Controllers, with the Model notifying all user interfaces about updates.

Model View Controller pattern makes it easier for the development of an application faster as it is easier for multiple developers to collaborate and work together. For example, in the context of Business Intelligence (BI), the Model View Controller allows for the creation of multiple views for a single model, enabling different stakeholders such as managers, analysts, and developers to have different representations of data formats, such as graphs and pie charts. [9] Although a slight disadvantage of the Model View Controller architecture is that it can be hard to understand because of the deployment of strict rules on methods.

2.5. Micro services

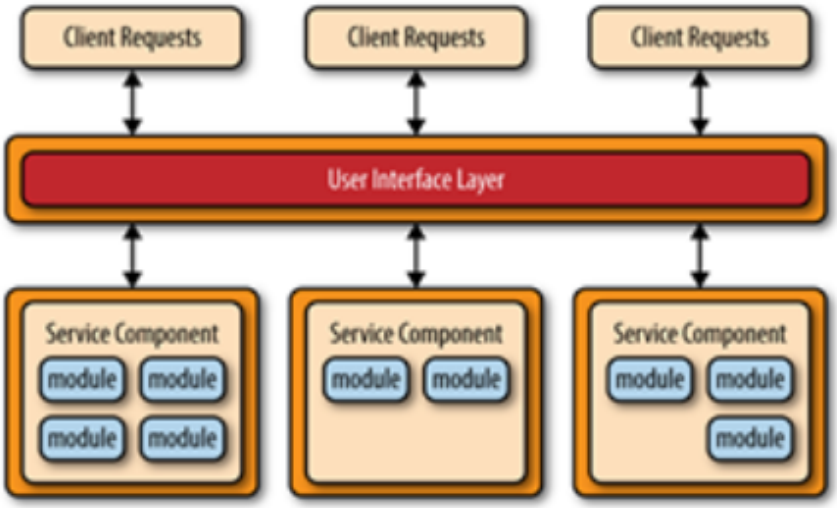

Figure 5: Micro-services.

An architectural style of micro-services (MSA) is an approach where applications are organized as small, independent applications with each running

its own processes and communication between them based on simple mechanisms like HTTP Resource API. This method allows for more flexible design options; however, it also possesses a number of technical and organizational challenges. The transition from a legacy to new architecture involves the following three main steps: re-engineering the existing system, the changeover between legacy architecture to new ones, and the implementation of the new system. Migrating from one system to another system demands careful preplanning and execution to handle technical as well as organisational issues. This research provides insights into labour migration challenges and workforce that can be helpful for both scholars and practitioners in this field.

The report presents the results of a research study, on how companies in the software field's adopting Micro-service Architectures (MSA). It involves a survey of 18 professionals engaged in transitioning existing systems to MSA. The research discussed in this article covers aspects of the transition process including recovering architecture making transformations and implementing the system. Participants have been incorporating features during the transition indicating an iterative approach to the process. The study also shows migration challenges like setting micro-services priorities sharing knowledge and creating cohesion across domains. The article gives importance to addressing both organizational hurdles during transitions. It shows insights into the duties and obstacles encountered by professionals offering insights, for researchers and industry experts.

This research highlights some actions, for professionals like sharing experiences making sure the alignment between business and IT and overseeing development efforts during this transition. It also gives much importance to the need to address the challenge of migrating existing databases to microservices as a core focus for academics. The primary goal of the study is to offer an approach to facilitate migration processes and present recommendations for future research and practical implementation in the industry. Additionally it examines the study's constraints, such, as its sample size. Proposes future endeavors to tackle these limitations and enhance the clarity of the results. [10]

3. Conclusion

S.No.	Pattern	Reusability	Flexibility	Performance and Efficiency	Scalability	Maintainability
1.	Client-Server	Strength	Strength	Neutral	Strength	Neutral
2.	Pipe and Filters	Strength	Strength	Neutral	Neutral	strength
3.	SOA	Strength	Strength	Strength	Strength	Strength
4.	MVC	Strength	Strength	Liabilities	Liabilities	Liabilities
5.	Micro-Services	Strength	Strength	Strength	Strength	Strength

Table 1: Tabular representation of Relationship between Architectural Pattern and Quality attributes. [1]

In the above table, we went through each paper of respective pattern and then classified the architectural pattern based on whether to utilize a pattern or to avoid it based on the architectural pattern relationship between pattern and quality attributes. In the above table, we can see that if we want scalability then using MVC is a liability which means we must try to avoid that architectural pattern, whereas using Microservice architecture has strength in scalability so the Micro-Service pattern can be used for scalability quality attribute. In this way, a systematic literature review of papers on different quality attributes helps the architect to decide on a suitable architectural pattern based on a particular quality attribute.

References

[1] S. Farshidi, S. Jansen, J. M. van der Werf, Capturing software architecture knowledge for pattern-driven design, Journal of Systems and Software 169 (2020) 110714.

[2] A. Sharma, M. Kumar, S. Agarwal, A complete survey on software architectural styles and patterns, Procedia Computer Science 70 (2015) 16–28.

[3] E. Majidi, M. Alemi, H. Rashidi, Software architecture: A survey and classification, in: 2010 Second International Conference on Communication Software and Networks, IEEE, 2010, pp. 454–460.

[4] H. S. Oluwatosin, Client-server model, IOSR Journal of Computer Engineering 16 (1) (2014) 67–71.

[5] C. Wulf, W. Hasselbring, J. Ohlemacher, Parallel and generic pipe-and-filter architectures with teetime, in: 2017 IEEE International Conference on Software Architecture Workshops (ICSAW), IEEE, 2017, pp. 290–293.

[6] M. Galster, P. Avgeriou, Qualitative analysis of the impact of soa patterns on quality attributes, in: 2012 12th international conference on quality software, IEEE, 2012, pp. 167–170.

[7] N. Niknejad, W. Ismail, I. Ghani, B. Nazari, M. Bahari, et al., Understanding service-oriented architecture (soa): A systematic literature review and directions for further investigation, Information Systems 91 (2020) 101491.

[8] A. Freeman, The MVC Pattern, Apress, Berkeley, CA, 2012, pp. 47–71. doi:10.1007/978-1-4302-4237-6_3.

[9] M. Kalelkar, P. P. Churi, D. Kalelkar, Implementation of model-view-controller architecture pattern for business intelligence architecture, International Journal of Computer Applications 102 (2014) 16–21.
URL `https://api.semanticscholar.org/CorpusID:3387026`

[10] P. Di Francesco, P. Lago, I. Malavolta, Migrating towards microservice architectures: an industrial survey, in: 2018 IEEE international conference on software architecture (ICSA), IEEE, 2018, pp. 29–2909.