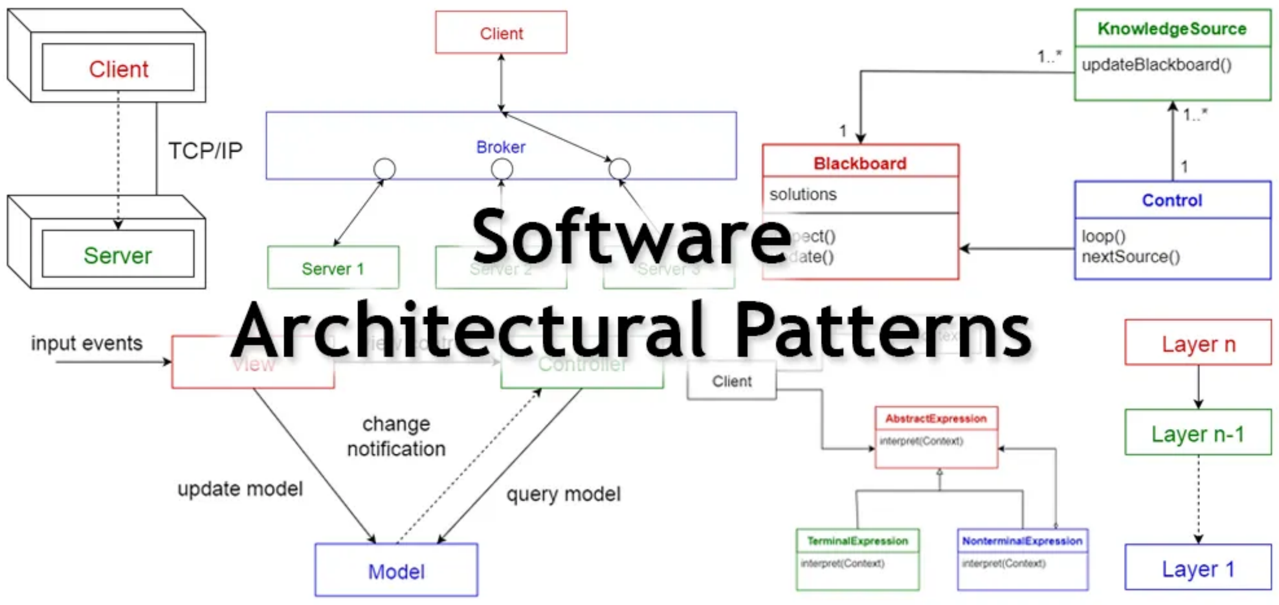
tds Published in Towards Data Science

Vijini Mallawaarachchi Follow
Sep 4, 2017 · 5 min read · ✦ · Listen

10 Common Software Architectural Patterns in a nutshell

Ever wondered how large enterprise scale systems are designed? Before major software development starts, we have to choose a suitable architecture that will provide us with the desired functionality and quality attributes. Hence, we should understand different architectures, before applying them to our design.



What is an Architectural Pattern?

According to Wikipedia

> *An **architectural pattern** ... nonly occurring problem in software architecture within a given context. Architectural patterns are similar to software design pattern but have a broader scope.*

In this article, I will be briefly explaining the following 10 common architectural patterns with their usage, pros and cons.

1. **Layered pattern**
2. **Client-server pattern**
3. **Master-slave pattern**
4. **Pipe-filter pattern**
5. **Broker pattern**
6. **Peer-to-peer pattern**
7. **Event-bus pattern**
8. **Model-view-controller pattern**
9. **Blackboard pattern**
10. **Interpreter pattern**

1. Layered pattern

This pattern can be used to structure programs that can be decomposed into groups of subtasks, each of which is at a particular level of abstraction. Each layer provides services to the next higher layer.

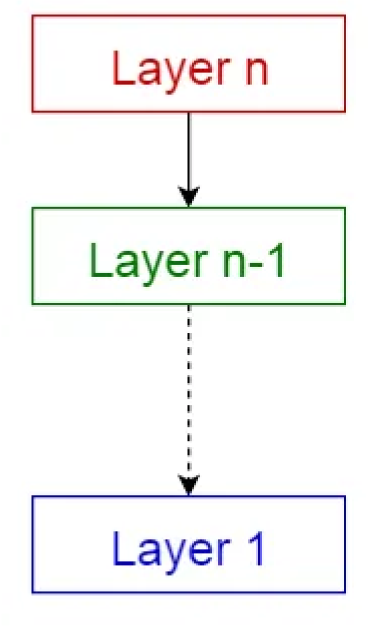
The most commonly found 4 layers of a general information system are as follows.

- **Presentation layer** (also known as **UI layer**)
- **Application layer** (also known as **service layer**)
- **Business logic layer** (also known as **domain layer**)

- **Data access layer** (also known as **persistence layer**)

Usage

- General desktop applications.
- E commerce web applications.



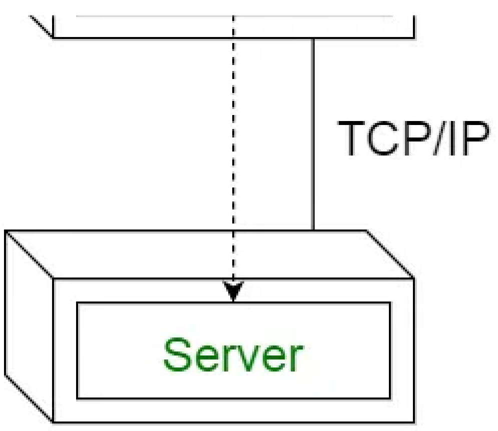
Layered pattern

2. Client-server pattern

This pattern consists of two parties; a **server** and multiple **clients**. The server component will provide services to multiple client components. Clients request services from the server and the server provides relevant services to those clients. Furthermore, the server continues to listen to client requests.

Usage

- Online applications such as email, document sharing and banking.



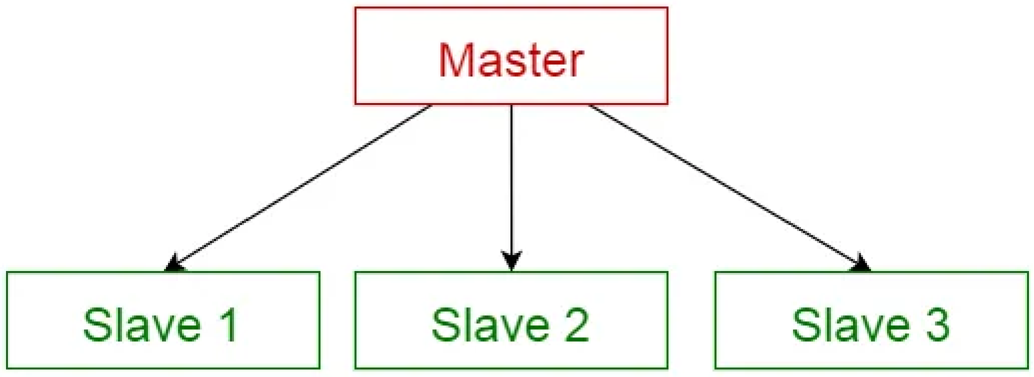
Client-server pattern

3. Master-slave pattern

This pattern consists of two parties; **master** and **slaves.** The master component distributes the work among identical slave components, and computes a final result from the results which the slaves return.

Usage

- In database replication, the master database is regarded as the authoritative source, and the slave databases are synchronized to it.
- Peripherals connected to a bus in a computer system (master and slave drives).



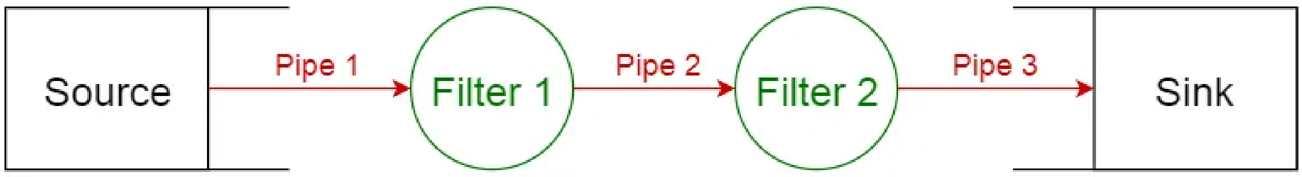
Master-slave pattern

4. Pipe-filter pattern

This pattern can be used to structure systems which produce and process a stream of data. Each processin onent. Data to be processed is passed thr or buffering or for synchronization purposes.

Usage

- Compilers. The consecutive filters perform lexical analysis, parsing, semantic analysis, and code generation.
- Workflows in bioinformatics.



Pipe-filter pattern

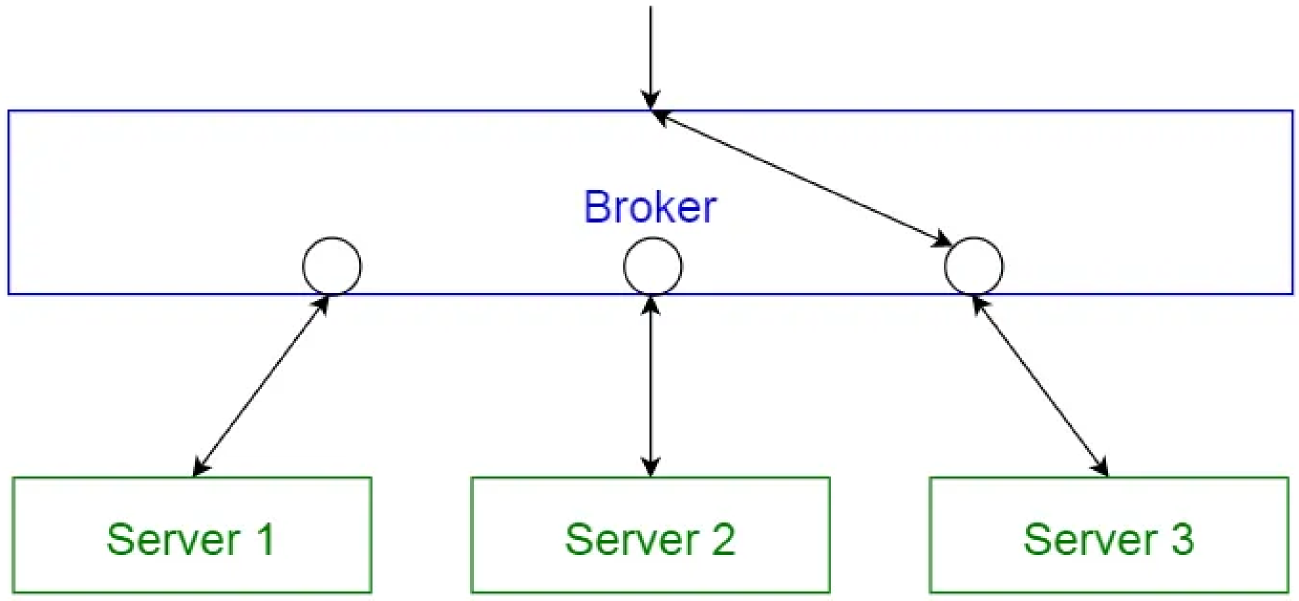
5. Broker pattern

This pattern is used to structure distributed systems with decoupled components. These components can interact with each other by remote service invocations. A **broker** component is responsible for the coordination of communication among **components**.

Servers publish their capabilities (services and characteristics) to a broker. Clients request a service from the broker, and the broker then redirects the client to a suitable service from its registry.

Usage

- Message broker software such as Apache ActiveMQ, Apache Kafka, RabbitMQ and JBoss Messaging.



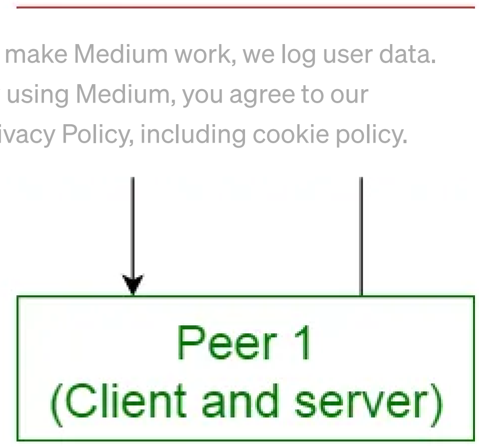
Broker pattern

6. Peer-to-peer pattern

In this pattern, individual components are known as **peers**. Peers may function both as a **client**, requesting services from other peers, and as a **server**, providing services to other peers. A peer may act as a client or as a server or as both, and it can change its role dynamically with time.

Usage

- File-sharing networks such as **Gnutella** and **G2**)
- Multimedia protocols such as **P2PTV** and **PDTP**.
- Cryptocurrency-based products such as **Bitcoin** and **Blockchain**



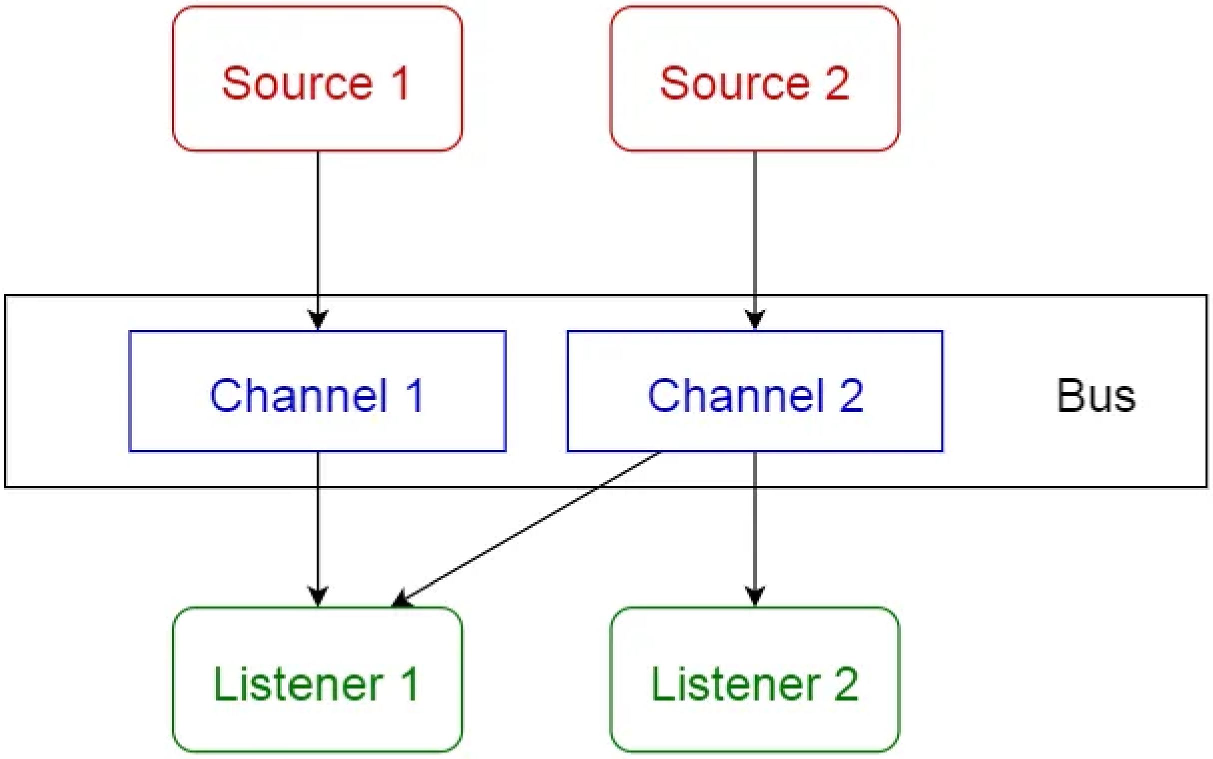
Peer-to-peer pattern

7. Event-bus pattern

This pattern primarily deals with events and has 4 major components; **event source**, **event listener**, **channel** and **event bus.** Sources publish messages to particular channels on an event bus. Listeners subscribe to particular channels. Listeners are notified of messages that are published to a channel to which they have subscribed before.

Usage

- Android development
- Notification services



Event-bus pattern

8. Model-view-contr

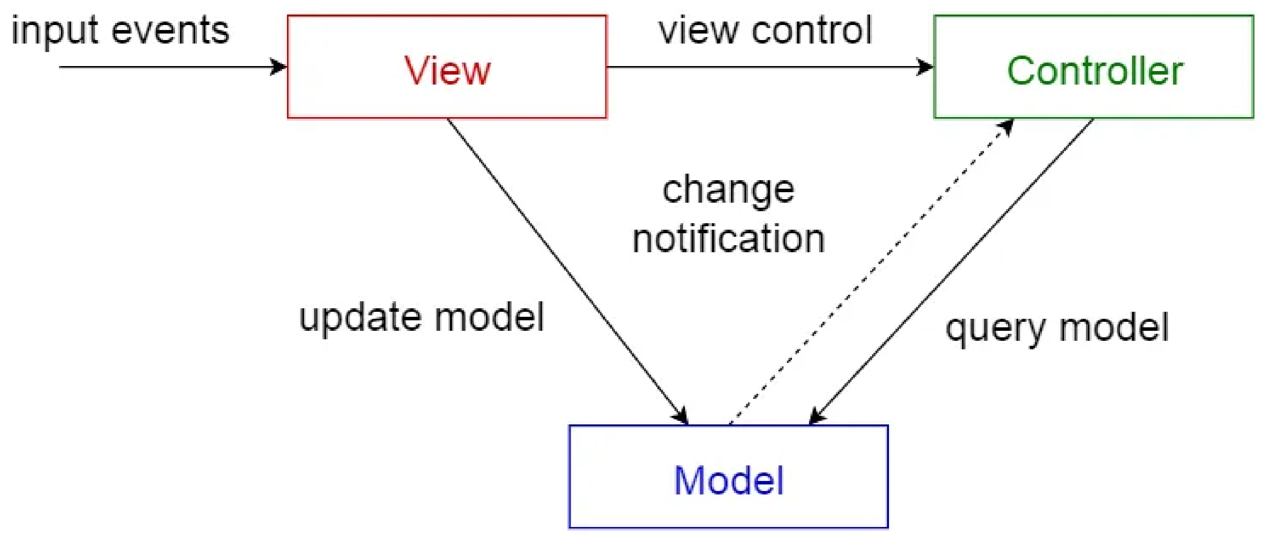
This pattern, also known as MVC pattern, divides an interactive application in to 3 parts as,

1. **model** — contains the core functionality and data
2. **view** — displays the information to the user (more than one view may be defined)
3. **controller** — handles the input from the user

This is done to separate internal representations of information from the ways information is presented to, and accepted from, the user. It decouples components and allows efficient code reuse.

Usage

- Architecture for World Wide Web applications in major programming languages.
- Web frameworks such as **Django** and **Rails**.



Model-view-controller pattern

9. Blackboard pattern

This pattern is useful for problems for which no deterministic solution strategies are known. The blackboard pattern consists of 3 main components.

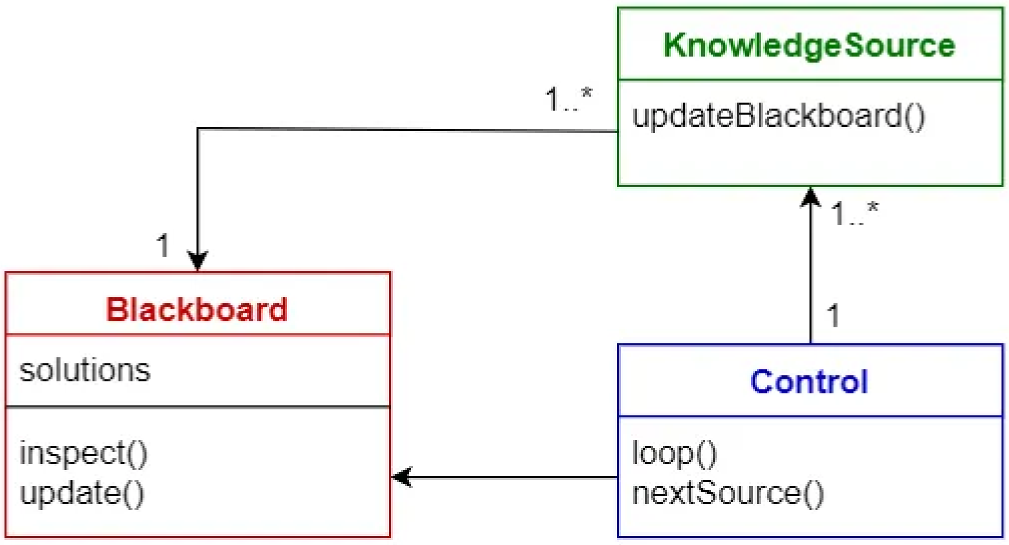
- **blackboard** — a stru ects from the solution space

- **knowledge source** — specialized modules with their own representation
- **control component** — selects, configures and executes modules.

All the components have access to the blackboard. Components may produce new data objects that are added to the blackboard. Components look for particular kinds of data on the blackboard, and may find these by pattern matching with the existing knowledge source.

Usage

- Speech recognition
- Vehicle identification and tracking
- Protein structure identification
- Sonar signals interpretation.



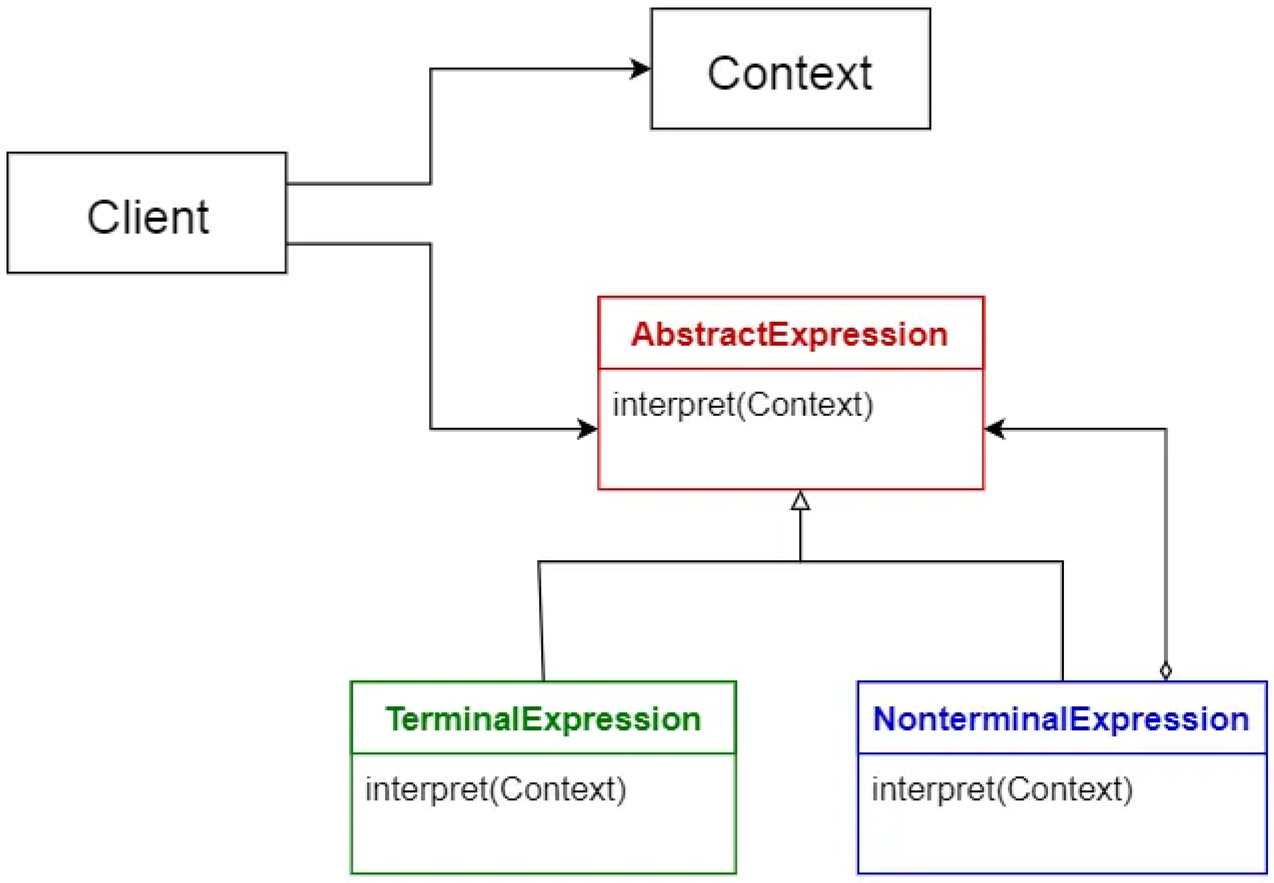
Blackboard pattern

10. Interpreter pattern

This pattern is used for designing a component that interprets programs written in a dedicated language. It mainly specifies how to evaluate lines of programs, known as sentences or expressions written in a particular language. The basic idea is to have a class for each symbol o

Usage

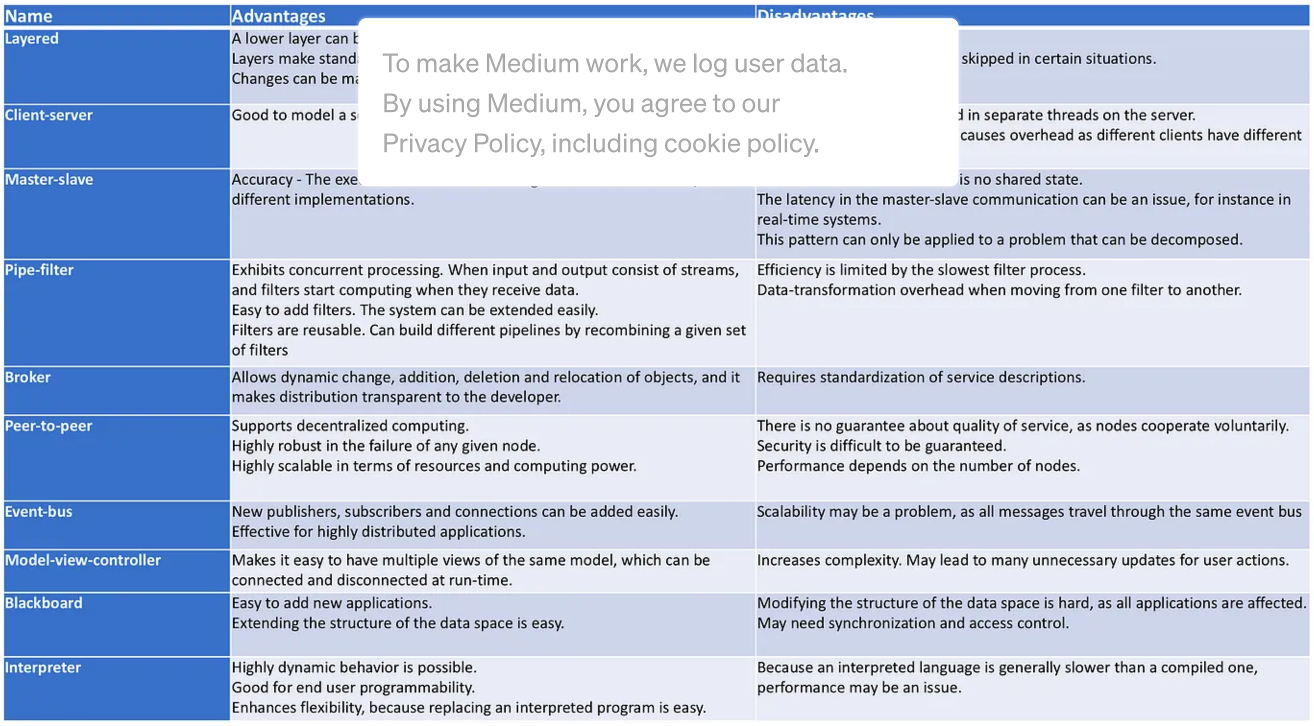
- Database query languages such as SQL.
- Languages used to describe communication protocols.



Interpreter pattern

Comparison of Architectural Patterns

The table given below summarizes the pros and cons of each architectural pattern.



Name	Advantages	Disadvantages
Layered	A lower layer can b Layers make stand Changes can be ma	skipped in certain situations.
Client-server	Good to model a s	d in separate threads on the server. causes overhead as different clients have different
Master-slave	Accuracy - The exe different implementations.	is no shared state. The latency in the master-slave communication can be an issue, for instance in real-time systems. This pattern can only be applied to a problem that can be decomposed.
Pipe-filter	Exhibits concurrent processing. When input and output consist of streams, and filters start computing when they receive data. Easy to add filters. The system can be extended easily. Filters are reusable. Can build different pipelines by recombining a given set of filters	Efficiency is limited by the slowest filter process. Data-transformation overhead when moving from one filter to another.
Broker	Allows dynamic change, addition, deletion and relocation of objects, and it makes distribution transparent to the developer.	Requires standardization of service descriptions.
Peer-to-peer	Supports decentralized computing. Highly robust in the failure of any given node. Highly scalable in terms of resources and computing power.	There is no guarantee about quality of service, as nodes cooperate voluntarily. Security is difficult to be guaranteed. Performance depends on the number of nodes.
Event-bus	New publishers, subscribers and connections can be added easily. Effective for highly distributed applications.	Scalability may be a problem, as all messages travel through the same event bus
Model-view-controller	Makes it easy to have multiple views of the same model, which can be connected and disconnected at run-time.	Increases complexity. May lead to many unnecessary updates for user actions.
Blackboard	Easy to add new applications. Extending the structure of the data space is easy.	Modifying the structure of the data space is hard, as all applications are affected. May need synchronization and access control.
Interpreter	Highly dynamic behavior is possible. Good for end user programmability. Enhances flexibility, because replacing an interpreted program is easy.	Because an interpreted language is generally slower than a compiled one, performance may be an issue.

Comparison of Architectural Patterns

Hope you found this article useful. I would love to hear your thoughts. 😇

Thanks for reading. 😊

Cheers! 😃

References

https://www.ou.nl/documents/40554/791670/IM0203_03.pdf/30dae517-691e-b3c7-22ed-a55ad27726d6

Software Architecture Programming Computer Science Software Engineering

Towards Data Science

Sign up for The Variable

By Towards Data Science

Every Thursday, the Variable deli ıds-on tutorials and cutting-edge research to original feature

Get this newsletter