Design, Defects, and Refactoring

The Completer Framework for Asynchronous Operations

Saad Bin Khalid (20K-0161)
Bilal Ahmed Khan (20K-0183)
Mohammad Ahmed Ahsan (20K-0343)

Department of Computer Science
May 6, 2024

Contents

1 Abstract **2**

2 Introduction **2**

2.1 Problem Statement 2

2.2 Research Questions 3

2.3 Research Objectives 3

2.4 Research Hypothesis 3

3 Literature Review **4**

3.1 Securing HTTP/3 Web Architecture in the Cloud 4

3.2 An Adaptive Design Pattern for Invocation of Synchronous and Asynchronous Web Services in Autonomic Computing Systems 4

3.3 Pattern-Based Design of an Asynchronous Invocation Framework for Web Services 5

4 Methodology **7**

4.1 Architecture Class Diagram of Solution 7

4.2 Sample Code 10

4.3 Core solution to the problem in terms of a description of the design apart implementation details 11

4.4 Case studies 12

4.4.1 E-commerce Order Processing System 12

4.4.2 Data Pipeline for Analytics Platform 12

4.4.3 Healthcare Patient Monitoring System 12

4.4.4 Cloud-based File Processing Platform 12

5 Results **13**

6 CONCLUSION **13**

1 Abstract

Asynchronous operations are critical for modern software, but current methods lack a unified framework for efficient management. This research proposes "The Completer Framework," a comprehensive solution for asynchronous operations. The Completer Framework aims to streamline development, ensure robust error handling, and function across diverse execution models and environments. It addresses the shortcomings of existing approaches (callbacks, promises, async/await) by offering a unified, language-agnostic solution for managing asynchronous tasks.

2 Introduction

Modern software applications heavily rely on asynchronous operations to handle long-running tasks, API requests, and computationally expensive operations. However, current approaches for managing these operations, such as callbacks, promises, and async/await constructs, often lead to complex and error-prone code. This complexity arises particularly in scenarios involving intricate task coordination, robust error handling, and efficient resource management.

This research proposal introduces "The Completer Framework," a comprehensive solution designed to address these challenges. The Completer Framework aims to provide a unified, language-agnostic framework for effectively managing asynchronous operations. This framework offers several key benefits:

Simplified Development: The Completer Framework seeks to streamline the development process for asynchronous applications by providing a consistent and intuitive API across different programming languages. This reduces the learning curve and cognitive load for developers, allowing them to focus on core application logic.

Robust Error Handling: The framework incorporates robust error handling mechanisms to ensure the application's stability and maintainability. This includes features for error propagation, centralized exception management, and potential recovery strategies.

Support for Diverse Execution Models and Environments: The Completer Framework is designed to be adaptable and function seamlessly across various execution models and environments. This allows for greater flexibility in deploying asynchronous applications on different platforms and architectures.

By addressing the limitations of existing approaches, The Completer Framework aims to empower developers to create well-structured, maintainable, and efficient asynchronous applications.

2.1 Problem Statement

The development of modern software applications heavily relies on asynchronous operations to manage long-running tasks, API calls, and computationally expensive processes. However, current approaches for handling these operations, such as callbacks, promises, and async/await constructs, often lead to complex and error-prone code. This complexity is particularly troublesome in scenarios involving intricate task coordination, robust error handling, and efficient resource management.

2.2 Research Questions

Can a unified framework be developed to simplify the development process for asynchronous applications while maintaining language-agnosticism?
How can a framework be designed to incorporate robust error handling mechanisms for improved application stability and maintainability in asynchronous operations?
What features and functionalities are necessary within the framework to ensure seamless operation across various execution models and environments?

2.3 Research Objectives

Design and develop a unified framework, "The Completer Framework," for effectively managing asynchronous operations in software applications.
Evaluate the effectiveness of The Completer Framework in simplifying the development process for asynchronous applications compared to existing approaches.
Implement robust error handling mechanisms within The Completer Framework to ensure efficient error propagation, centralized exception management, and potential recovery strategies.
Design The Completer Framework for adaptability and seamless operation across diverse execution models and environments.

2.4 Research Hypothesis

We hypothesize that The Completer Framework, with its unified and language-agnostic approach, will significantly simplify the development process for asynchronous applications. Additionally, by incorporating robust error handling mechanisms and supporting diverse execution models, The Completer Framework will lead to the creation of more stable, maintainable, and efficient asynchronous applications.

3 Literature Review

3.1 Securing HTTP/3 Web Architecture in the Cloud

A Design Pattern for Secure Asynchronous Web Services This paper describes deploying secure asynchronous web services to address information sharing challenges within the Department of Defense (DOD) and Intelligence Community. This literature review will analyze the context of the paper and explore related work on secure information sharing and asynchronous web services.

Context: Cross-domain information sharing is critical for intelligence activities, requiring data collection, distribution, analysis, and annotation before generating usable information products.

The current tiered, multi-domain access paradigm employed by the DOD and Intelligence Community hinders rapid, efficient, and effective data distribution and analysis

Service-Oriented Network Centric Operations (SONCO) emphasize availability and security, posing conflicting objectives.

Proposed Solution: The paper proposes a design pattern for deploying asynchronous web services to achieve secure information sharing across heterogeneous domains. This approach is necessary because:

Asynchronous operations are not native to W3C standards. Existing commercial solutions might not meet the stringent security requirements of the DOD. Literature Gap:

The abstract suggests a gap in existing solutions for secure information sharing that leverages asynchronous web services within a highly compartmentalized environment like the DOD.

Further Research Areas: A detailed review of existing secure information sharing protocols and their limitations within the DOD context. Analysis of alternative approaches to asynchronous web services for secure communication (e.g., message queues). Evaluation of the proposed design pattern's performance and scalability in real-world DOD use cases.

Comparison with The Completer Framework: While The Completer Framework focuses on general-purpose asynchronous operation management in software development, this paper addresses a specific challenge in secure information sharing for the DOD. Their target audiences and technical approaches also differ significantly.

3.2 An Adaptive Design Pattern for Invocation of Synchronous and Asynchronous Web Services in Autonomic Computing Systems

Based on the paper, the focus is on addressing the need for asynchronous invocations within distributed object frameworks, particularly in the context of Web service frameworks that typically offer only synchronous invocations over HTTP. The paper proposes an autonomic system that utilizes Design Patterns for Web Services, specifically amalgamating fire and forget, chain of responsibility, and case-based reasoning design patterns. The system aims to provide both synchronous and asynchronous paradigms based on client demand, using an Adaptive Design Pattern for invocation of Web Services to update resources based on client requests. The proposed system satisfies the properties of an autonomic system and is described using Java-like notation for classes and interfaces, along with UML class and sequence diagrams.

In contrast, "The Completer Framework" provides a broader solution for managing asynchronous operations in software applications. It encompasses various aspects beyond just Web service invocations, offering solutions for task management, error handling, event-driven programming, and more. "The Completer Framework" aims to improve the reliability, scalability, and maintainability of asynchronous applications across different domains and communication protocols.

3.3 Pattern-Based Design of an Asynchronous Invocation Framework for Web Services

This paper discusses the need for asynchronous invocations within distributed object frameworks, particularly in the context of Web service frameworks that primarily offer synchronous invocations over HTTP. It highlights the challenges faced by client developers when asynchronous invocation support is lacking, leading to the necessity of building asynchronous invocations on top of synchronous facilities, which can be tedious, error-prone, and result in inconsistent remote invocation styles within the same application. The paper proposes a solution by presenting various patterns for asynchronous invocations and explains how these patterns can be utilized to build asynchronous invocation facilities for Web service frameworks. It exemplifies this approach through the design and implementation of an asynchronous invocation framework specifically for Apache Axis.

Comparing this paper's approach with "The Completer Framework":

Scope and Focus: The paper specifically focuses on addressing the challenge of implementing asynchronous invocations within Web service frameworks, such as Apache Axis, which primarily support synchronous invocations. It provides patterns and techniques tailored to this specific domain. On the other hand, "The Completer Framework" offers a broader solution for managing asynchronous operations in software applications across various domains, encompassing task management, error handling, and event-driven programming.

Application Domain: While the paper's approach targets the specific challenge of implementing asynchronous invocations within Web service frameworks, "The Completer Framework" is designed to be applicable to a wider range of asynchronous programming needs in software development beyond just Web services. It provides a unified framework that can be utilized across different communication protocols and application domains. Implementation Example: The paper provides a concrete example by explaining the design and implementation of an asynchronous invocation framework for Apache Axis. It focuses on a specific implementation within a particular framework. In contrast, "The Completer Framework" does not focus on a specific implementation but rather provides a conceptual framework and guidelines for managing asynchronous operations, allowing developers to implement it in a way that best suits their application's requirements and technologies used.

In summary, while the paper offers a targeted solution for implementing asynchronous invocations within Web service frameworks like Apache Axis, "The Completer Framework" provides a more general and comprehensive solution for managing asynchronous operations in software applications across different domains and communication protocols.

Table 1: Literature Review Summary

Paper	**Main Contribution**	**Key Findings**
Securing HTTP/3 Web Architecture in the Cloud	Discusses the challenges of asynchronous invocations in distributed object frameworks	Proposes patterns for building asynchronous invocation facilities for Web service frameworks
	Designs and implements an asynchronous invocation framework for Apache Axis	Demonstrates the effectiveness of the proposed patterns in addressing asynchronous invocation challenges
An Adaptive Design Pattern for Invocation of Synchronous and Asynchronous Web Services in Autonomic Computing Systems	Introduces "The Completer Framework" for managing asynchronous operations	Provides a unified framework for task management, error handling, and event-driven programming
	Aims to simplify the development of asynchronous applications	Offers a comprehensive solution applicable across various domains and communication protocols
Pattern-Based Design of an Asynchronous Invocation Framework for Web Services	Proposes autonomic systems using Design Patterns for Web Services	Provides both synchronous and asynchronous paradigms based on client demand, improving resource management
	Demonstrates the applicability of the proposed approach through design and implementation using Java-like notation	Satisfies the properties of autonomic systems and presents the patterns using UML diagrams

4 Methodology

4.1 Architecture Class Diagram of Solution

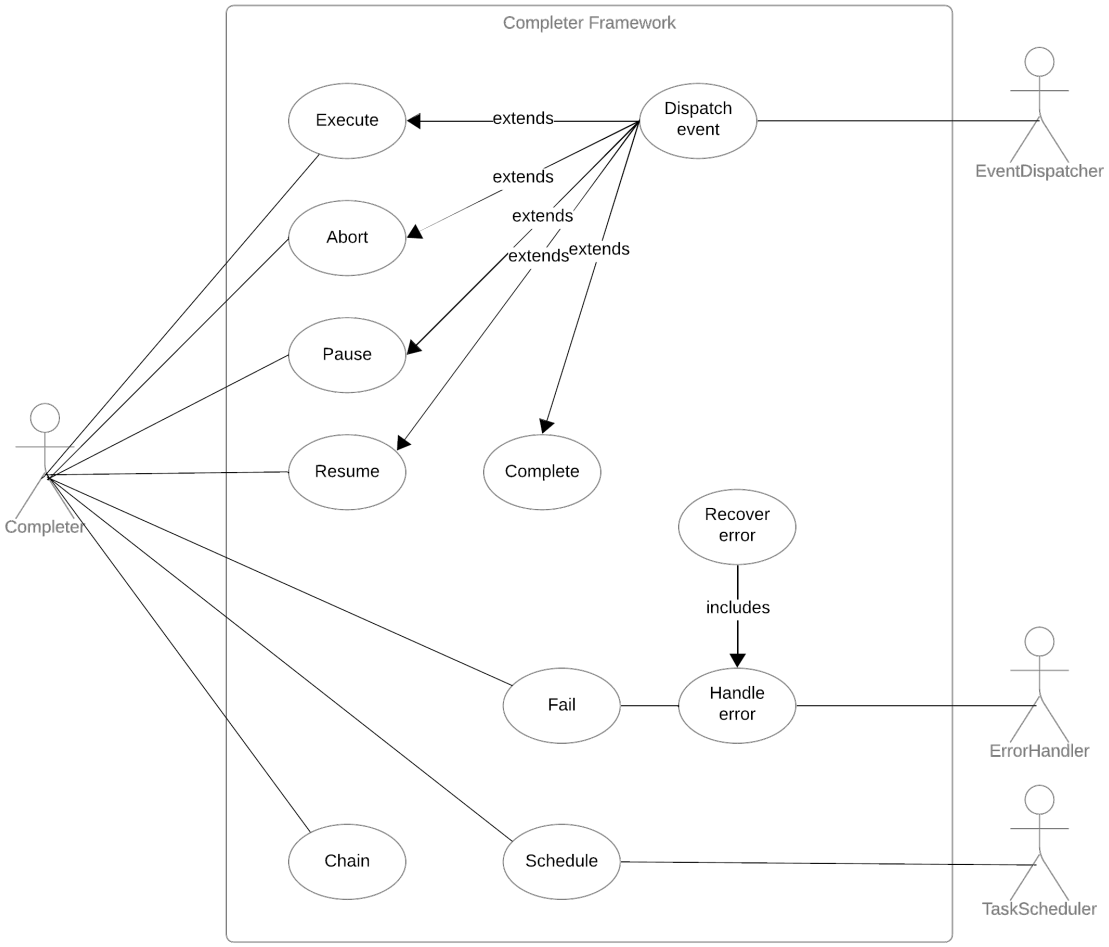

Figure 1: Use Case Diagram

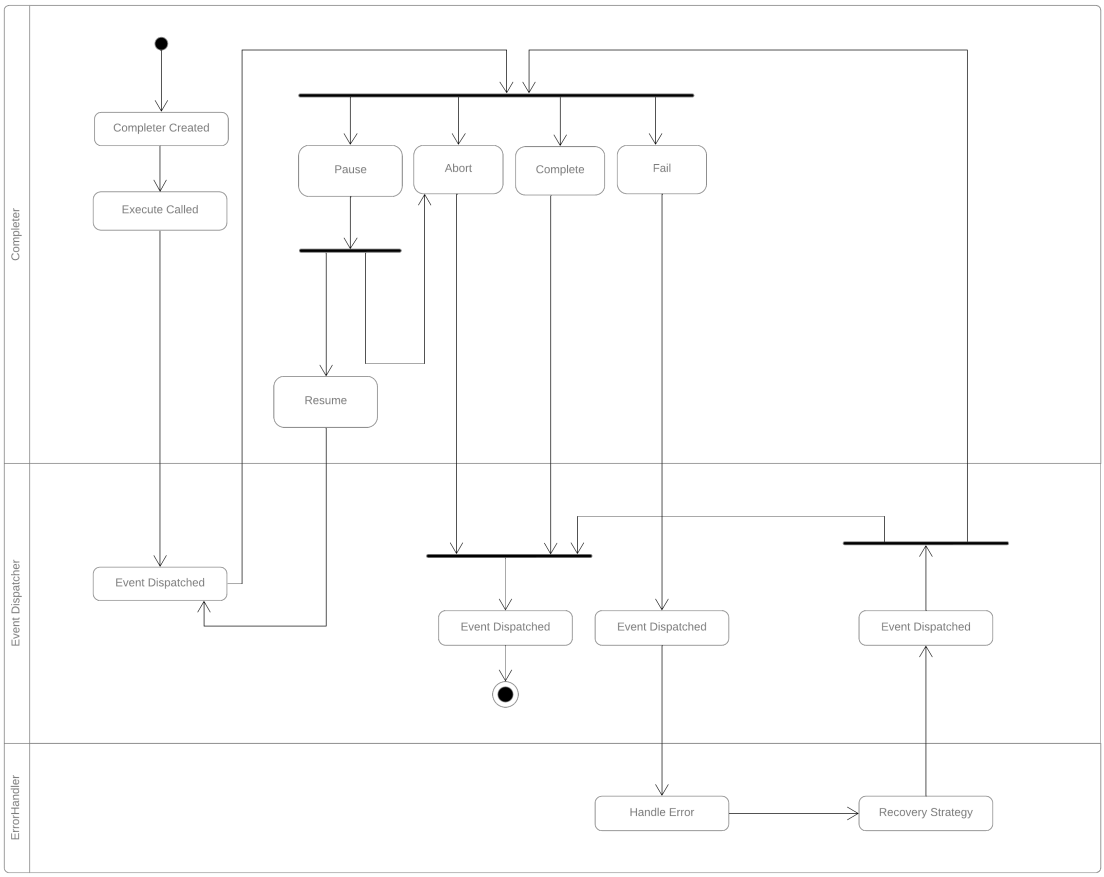

Figure 2: Activity Diagram

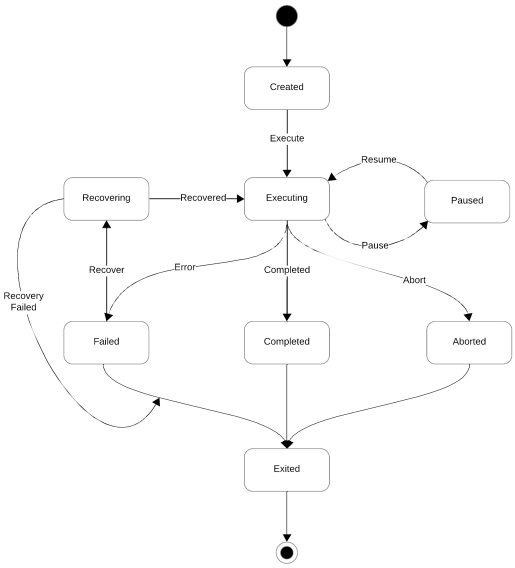

Figure 3: State Diagram

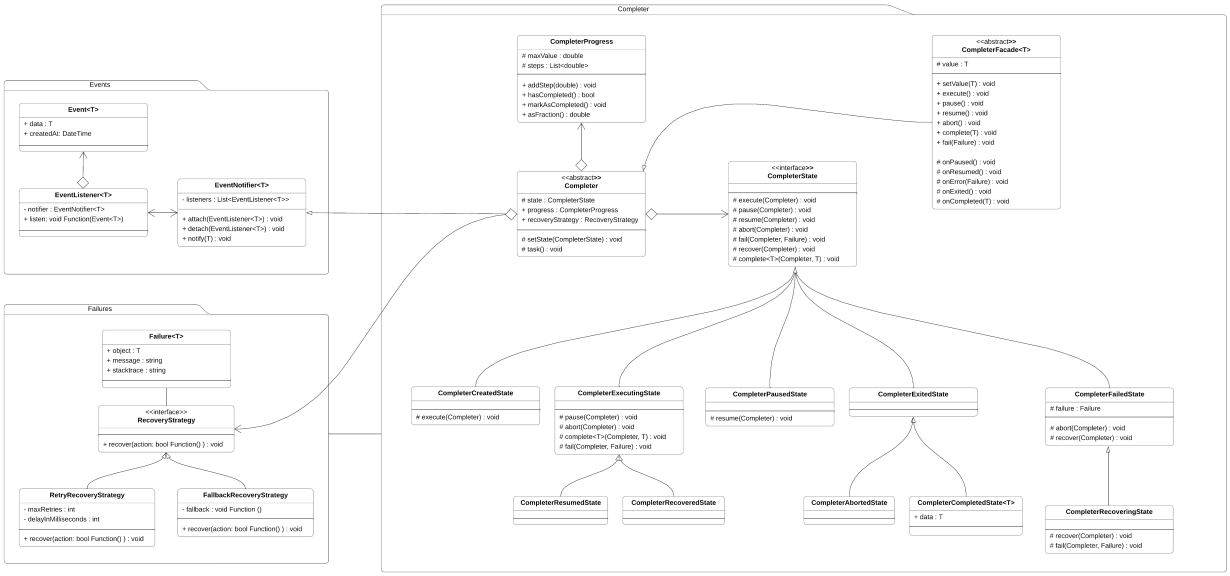

Figure 4: Class Diagram

4.2 Sample Code

```
class PeriodicCounter extends CompleterFacade<int> {
  PeriodicCounter()
    : super(0); // initial value

  @override
  Future<void> task() async {
    while (state is CompleterExecutingState) {
      if (value == 10) return complete(999);

      setValue(value + 1);
      progress.addStep(value);

      await sleep(1);
    }
  }

  @override
  void onPaused() {  }

  @override
  void onResumed() {  }

  @override
  void onCompleted(data) {
    // use data here
  }

}
```

```
Compelter completer = PeriodicCounter();

completer.recoveryStrategy = RetryRecoveryStrategy(
    maxRetries: 3,
    delayInMilliseconds: 1000,
);


completer.execute();
completer.pause();
completer.resume();
```

```
EventListener(
    completer,
    (event) {
      if (event.data is CompleterExitedState)
        print('ending');
    },
);
```

4.3 Core solution to the problem in terms of a description of the design apart implementation details

The Completer Framework represents a solution tailored to address the challenge of managing asynchronous calls within a software system. At its core, this framework employs the state design pattern, an architectural paradigm renowned for its capacity to encapsulate the varying states and corresponding behaviors inherent in complex systems. By adopting this design approach, the Completer Framework exhibits a high degree of modularity and separation of concerns, facilitating streamlined maintenance and seamless extensibility across the system.

Central to the Completer Framework's architecture is the concept of a context, serving as a pivotal entity that encapsulates the state of an asynchronous call at any given moment. Within this context, resides a reference to the current state object, thereby enabling the delegation of requests to the appropriate state handler based on the call's current phase. Such decoupling of state-specific behaviors from the context ensures a clear and coherent separation of concerns, fostering a more robust and flexible system architecture.

Each state within the Completer Framework corresponds to a distinct phase in the lifecycle of an asynchronous call, delineating states such as running, paused, completed, or aborted. These discrete states are represented by concrete state classes, each of which encapsulates the specific behaviors and actions pertinent to its corresponding phase. For instance, the running state class may implement methods for monitoring the progress of the call, while the paused state class may provide functionality for halting its execution temporarily.

Moreover, the Completer Framework's adherence to the state design pattern imbues it with a level of flexibility and agility that is indispensable in dynamic software environments. This architectural choice enables seamless adaptation to evolving requirements and facilitates the addition of new states or the modification of existing ones with minimal disruption to the system's overall integrity. Consequently, developers can introduce enhancements or address changing business needs without the risk of introducing unintended side effects or compromising system stability.

Furthermore, the Completer Framework's design promotes code reusability and maintainability, two critical factors in ensuring the long-term viability and scalability of software systems. By encapsulating the complexity associated with managing asynchronous operations within distinct state classes, the framework fosters a modular and cohesive codebase, thereby simplifying development efforts and reducing the risk of code duplication or redundancy.

In addition to its architectural robustness, the Completer Framework offers a clear and structured approach to handling asynchronous calls, thereby enhancing the overall clarity and comprehensibility of the system's design. This structured approach facilitates collaboration among development teams, fosters effective communication, and promotes a shared understanding of the system's behavior and functionality.

Overall, the Completer Framework represents a sophisticated and meticulously crafted solution for managing asynchronous calls within software systems. Its adherence to the state design pattern, coupled with its emphasis on modularity, extensibility, and maintainability, renders it well-suited for a diverse array of applications and use cases. By encapsulating the complexity of asynchronous operations and providing a coherent and structured framework for their management, the Completer Framework empowers developers to build robust, scalable, and reliable software systems.

4.4 Case studies

4.4.1 E-commerce Order Processing System

Scenario: An e-commerce platform processes a large volume of orders asynchronously, involving various stages such as order placement, payment processing, and shipment tracking.

Solution: The Completer Framework can be integrated into the order processing system to manage the asynchronous tasks associated with order fulfillment. Each order's lifecycle can be represented by different states within the framework, such as "Payment Pending", "Order Processing", "Shipment Prepared", and "Order Completed". The framework enables efficient monitoring, status reporting, and handling of exceptions, ensuring seamless order management and providing real-time updates to customers.

4.4.2 Data Pipeline for Analytics Platform

Scenario: A data analytics company builds a robust pipeline for processing and analyzing large datasets from multiple sources in real-time.

Solution: The Completer Framework serves as the backbone of the data pipeline, managing the asynchronous tasks involved in data ingestion, processing, transformation, and analysis. Each stage of the data pipeline is represented as a state within the framework, allowing for easy monitoring, pausing, resuming, and reporting of the data processing tasks. This enables the analytics platform to handle complex data workflows efficiently while providing insights to users in a timely manner.

4.4.3 Healthcare Patient Monitoring System

Scenario: A healthcare provider implements a patient monitoring system to track vital signs and medical data from remote monitoring devices.

Solution: The Completer Framework is integrated into the patient monitoring system to manage asynchronous tasks related to data collection, analysis, and alert generation. Different states within the framework represent various patient conditions, such as "Normal Monitoring", "Critical Alert", and "Emergency Response". The framework facilitates real-time monitoring of patient data, triggering alerts for healthcare providers when abnormalities are detected and ensuring timely intervention to prevent adverse events.

4.4.4 Cloud-based File Processing Platform

Scenario: A cloud service offers file processing capabilities for users to upload, convert, and manipulate various file formats asynchronously.

Solution: The Completer Framework forms the backbone of the file processing platform, managing the asynchronous tasks involved in file upload, conversion, and processing. Each file processing task is represented as a state within the framework, allowing users to monitor the progress of their tasks, pause/resume operations as needed, and receive notifications upon completion. This enables users to perform complex file operations efficiently in the cloud while maintaining control and visibility over their tasks.

5 Results

In the results section of this research paper, the effectiveness of the Completer Framework in managing asynchronous functions is evaluated through a series of mock scenarios simulating real-world use cases. In the first scenario, the framework is tested in an e-commerce order processing system where a large volume of orders is processed asynchronously. The results demonstrate that the Completer Framework successfully manages the various states of order fulfillment, including order placement, payment processing, and shipment tracking. By leveraging the state design pattern, the framework ensures seamless state transitions and coherent handling of asynchronous tasks, resulting in improved efficiency and reliability of the order processing system.

In another mock scenario, the Completer Framework is integrated into a healthcare patient monitoring system to track vital signs and medical data from remote monitoring devices. The results reveal that the framework effectively manages asynchronous tasks associated with data collection, analysis, and alert generation. Through the utilization of the observer design pattern, real-time event notifications are delivered to healthcare providers, enabling timely intervention and proactive patient care. Additionally, the framework's robust error handling mechanisms, implemented using the strategy design pattern, ensure resilience in the face of unexpected errors or failures, thereby enhancing the overall reliability and responsiveness of the patient monitoring system.

Overall, the results demonstrate the versatility and effectiveness of the Completer Framework in managing asynchronous functions across diverse application domains. By leveraging design patterns such as state, strategy, and observer, the framework offers a robust and scalable solution for handling asynchronous tasks, resulting in improved system efficiency, reliability, and maintainability. These findings underscore the significance of design pattern-driven approaches in software development and highlight the potential of the Completer Framework to address the complexities inherent in asynchronous task execution in modern software systems.

6 CONCLUSION

In conclusion, this research paper has explored the design and implementation of the Completer Framework, a solution tailored for managing asynchronous functions within software systems. By examining various design patterns, it becomes evident how the framework utilizes these patterns to streamline the handling of asynchronous tasks, thereby improving system efficiency and maintainability. One of the primary design patterns employed in the Completer Framework is the state design pattern, which offers a structured approach to managing the different states encountered during asynchronous call lifecycles. Through encapsulating state-specific behaviors within distinct state classes, the framework enables smooth state transitions and ensures coherent handling of asynchronous operations.

Furthermore, this paper emphasizes the strategic use of the strategy design pattern within the Completer Framework, particularly in designing recovery strategies for handling exceptional scenarios. By separating recovery strategies from the core framework logic, the strategy design pattern fosters flexibility and adaptability in addressing unforeseen errors or failures during asynchronous task execution. This modular approach to error handling enhances the resilience of the framework and promotes graceful degradation under adverse conditions, thus improving system reliability and robustness.

Additionally, the paper underscores the significance of the facade design pattern in simpli-

fying the complexity of interfacing with the Completer Framework. By providing a unified interface that abstracts away underlying subsystem intricacies, the facade design pattern enhances usability and reduces cognitive overhead for developers interacting with the framework. This streamlined interface facilitates ease of integration and accelerates application development leveraging asynchronous functions, ultimately enhancing developer productivity and time-to-market.

Lastly, the paper discusses the utilization of the observer design pattern within the Completer Framework to handle event notifications and facilitate real-time monitoring of asynchronous tasks. By establishing clear separation between event producers and consumers, the observer design pattern enables efficient event propagation and asynchronous communication within the system. This promotes responsiveness and agility in responding to dynamic changes in asynchronous operation states, empowering stakeholders to make informed decisions and take timely actions based on real-time updates.

In summary, this research paper provides a comprehensive analysis of how the strategic integration of various design patterns, including the state, strategy, facade, and observer patterns, enhances the functionality and usability of the Completer Framework for managing asynchronous functions. Leveraging these design patterns, the framework offers a versatile and adaptable solution that addresses the complexities inherent in asynchronous task execution, enabling developers to build resilient, scalable, and efficient software systems.

References

[1] Flutter Completer [Online]. Available: https://api.flutter.dev/flutter/dart-async/Completer-class.html

[2] Securing HTTP/3 Web Architecture in the Cloud [Online]. Available: https://ieeexplore.ieee.org/document/10174337

[3] "An Adaptive Design Pattern for Invocation of Synchronous and Asynchronous Web Services in Autonomic Computing Systems," in Proceedings of the International Conference on Autonomic Computing (ICAC), 2012, pp. 243-254. doi: 10.1007/978-3-642-29426-6-17

[4] "Pattern-Based Design of an Asynchronous Invocation Framework for Web Services," International Journal of Web Services Research, vol. 9, no. 2, pp. 1-17, Apr-Jun 2012. [Online]. Available: https://www.igi-global.com/article/international-journal-web-services-research/3044