King Fahd University of Petroleum and Minerals

**Cloud Applications Engineering  
  
Section#1**

**Team#2**

**Ibrahim Almoajel**

**Mohammed Cadura**

**Mazen Alahmari**

**Ali Alrebdi**

**Design Pattern: Asynchronous Request-Reply**

**Design Pattern: Asynchronous Request-Reply**

**Context and Problem (Purpose of the Design Pattern)** Sometimes, a system sends a request to another service, but waiting for an immediate response would slow things down. This pattern helps by letting the system continue working while the response is processed in the background. It’s useful for handling long tasks, improving efficiency, and keeping things smooth.

**Explanation of the Design Pattern** Instead of waiting for a reply right away, the system sends a request and moves on. The service processes it in the background and sends back a response later. The reply can be received through a callback, a message queue, or by checking for updates (polling). This keeps things moving without unnecessary delays.

**When to Use the Design Pattern**

* When a task takes a long time, and the system shouldn’t be stuck waiting.
* In microservices or cloud-based applications where services talk to each other.
* When integrating with third-party services that may have slow responses.
* To improve user experience by avoiding freezing or delays in apps.
* In event-driven systems where actions happen based on triggers.

**When Not to Use the Design Pattern**

* When an immediate response is required.
* If using asynchronous processing would overcomplicate things.
* When your system doesn’t support message queues or event-driven communication.
* If delays in getting a response would cause issues.
* When a simple request-response setup is enough.

**Issues and Considerations**

* **Matching Requests and Replies:** Ensuring that responses go to the right requests.
* **Handling Errors:** Setting up retries or fallback options when things go wrong.
* **Scalability:** Making sure the system can handle many requests efficiently.
* **Timeouts:** Defining how long to wait before giving up on a response.
* **Security:** Keeping data safe when using queues or event-driven messaging.

**The Design Pattern Addresses Which Pillars of the Well-Architected Framework**

1. **Operational Excellence:** Keeps the system running efficiently without delays.
2. **Security:** Reduces direct exposure between services, improving protection.
3. **Reliability:** Helps recover from failures and manage errors properly.
4. **Performance Efficiency:** Prevents unnecessary waiting and improves speed.
5. **Cost Optimization:** Ensures resources are used effectively, reducing waste.