Simplified Documentation on Distributed Systems: Broker Pattern

From Mud to Structure

This category helps manage complex systems by breaking down a large task into smaller, cooperative subtasks. The goal is to avoid a system overwhelmed by numerous components.

- **Patterns**:
    - **Layers Pattern**: Organizes the system into layers.
    - **Pipes and Filters Pattern**: Breaks down tasks into sequential steps.
    - **Blackboard Pattern**: Multiple subsystems contribute to a solution.

Distributed Systems

In distributed systems, components are decoupled and communicate through intermediaries. The **Broker Pattern** provides a framework for distributed systems where independent components interact remotely.

Broker Pattern

The **Broker Pattern** allows components of a distributed system to communicate without needing to know the details of the other components' locations or interfaces. The broker coordinates all communication, forwarding requests, results, and exceptions.

Context

- A distributed and possibly heterogeneous system with independent and cooperating components.

Problem

When building complex distributed systems, allowing components to handle communication themselves can create dependency issues and limit the system to specific technologies. These systems also need dynamic service management like adding, removing, or locating components without compromising portability.

Solution

A broker component is introduced to manage communication. Servers register their services with the broker, and clients request these services through the broker. The broker forwards client requests to the appropriate server, and then returns the results.

How the Broker Pattern Works

1. **Clients** request services from the broker.
2. The **broker** forwards the request to the appropriate **server**.
3. The **server** executes the request and returns results to the broker.
4. The broker transmits the results back to the client.

- **Components of the Broker Pattern**:
    - **Clients**: Applications that request services.
    - **Servers**: Applications that provide services.
    - **Brokers**: Components that manage communication between clients and servers.
    - **Proxies**: Client-side and server-side proxies that handle message transmission.
    - **Bridges**: Handle communication between brokers in different networks.

Example Scenario

Imagine a web browser (client) requesting a webpage from a web server (server). Instead of the client directly managing communication with the server, the request goes through an internet broker. The broker locates the server, retrieves the webpage, and sends it back to the client. This decouples the client from the server, ensuring flexibility and scalability.

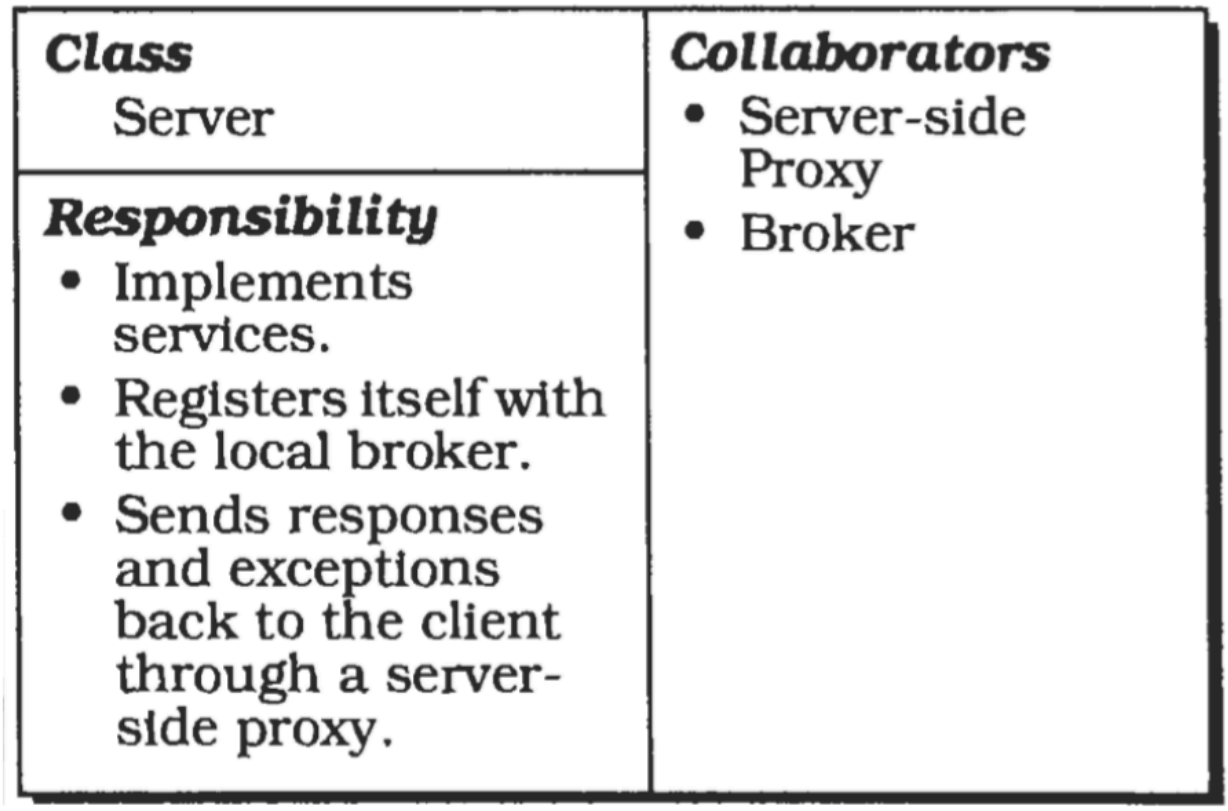
Diagram: Client-Broker-Server Communication (insert diagram here)

Components of the Broker Pattern

1. Servers

Servers implement objects that expose their functionality through interfaces, either via an interface definition language (IDL) or a binary standard. They can offer general services for many applications or domain-specific functionality.

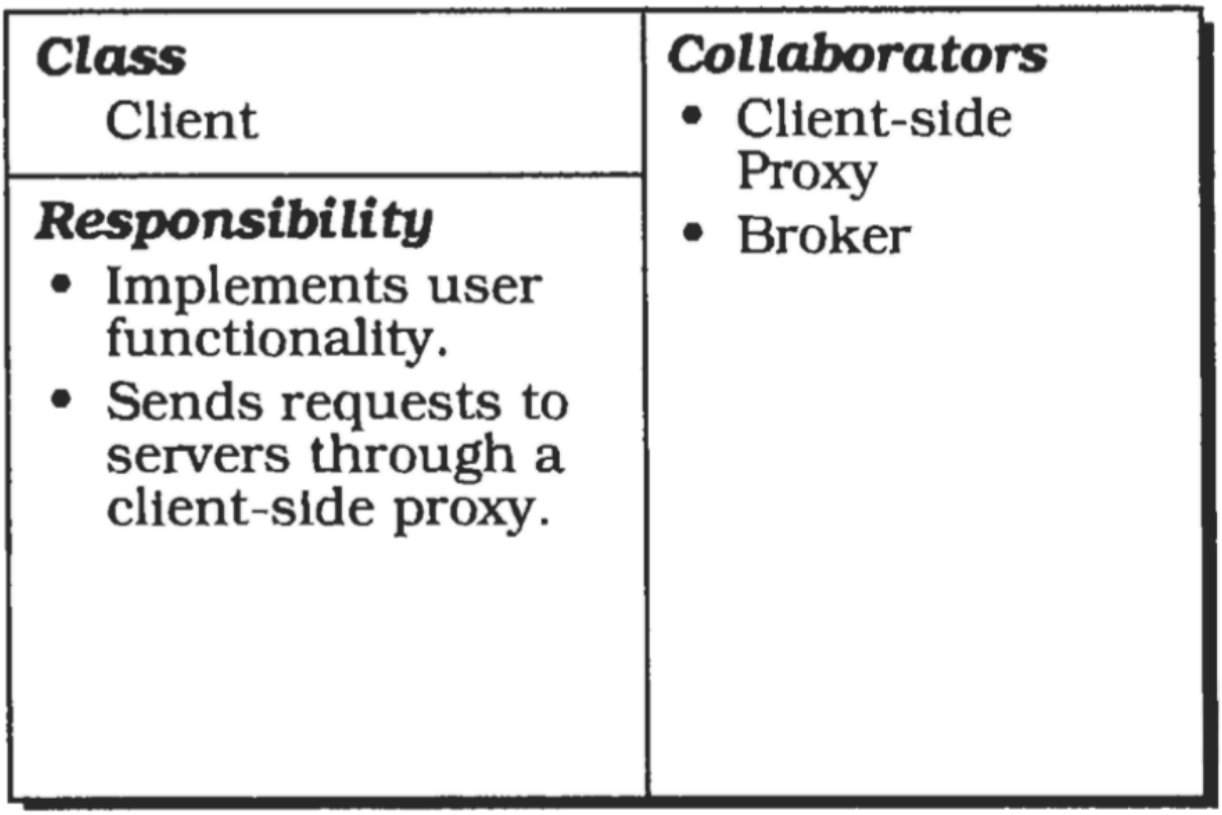
Diagram: CRC Diagram for Server



2. Clients

Clients request services from the broker. They don’t need to know the location of the servers providing the services, allowing services to be added, changed, or moved dynamically without affecting the system.

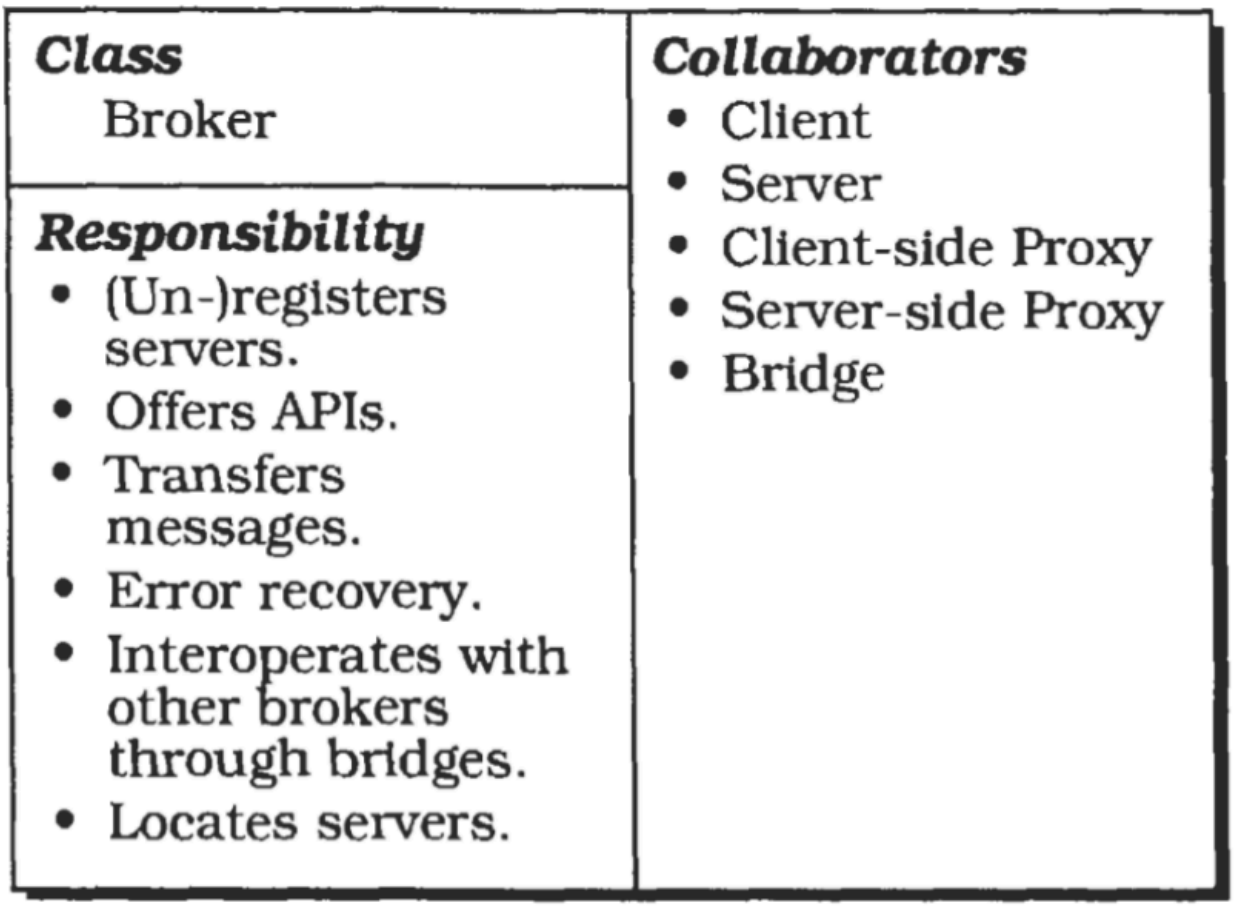
Diagram: CRC Diagram for Client



3. Brokers

Brokers are responsible for forwarding client requests to the correct servers and returning the results. If the requested server is inactive, the broker activates it. Brokers also handle communication between different brokers (if needed).

Diagram: CRC Diagram for Broker



4. Proxies

Proxies act as intermediaries between clients and servers. They hide the implementation details, manage communication, and handle marshalling/unmarshalling of data (converting data into formats suitable for transmission).

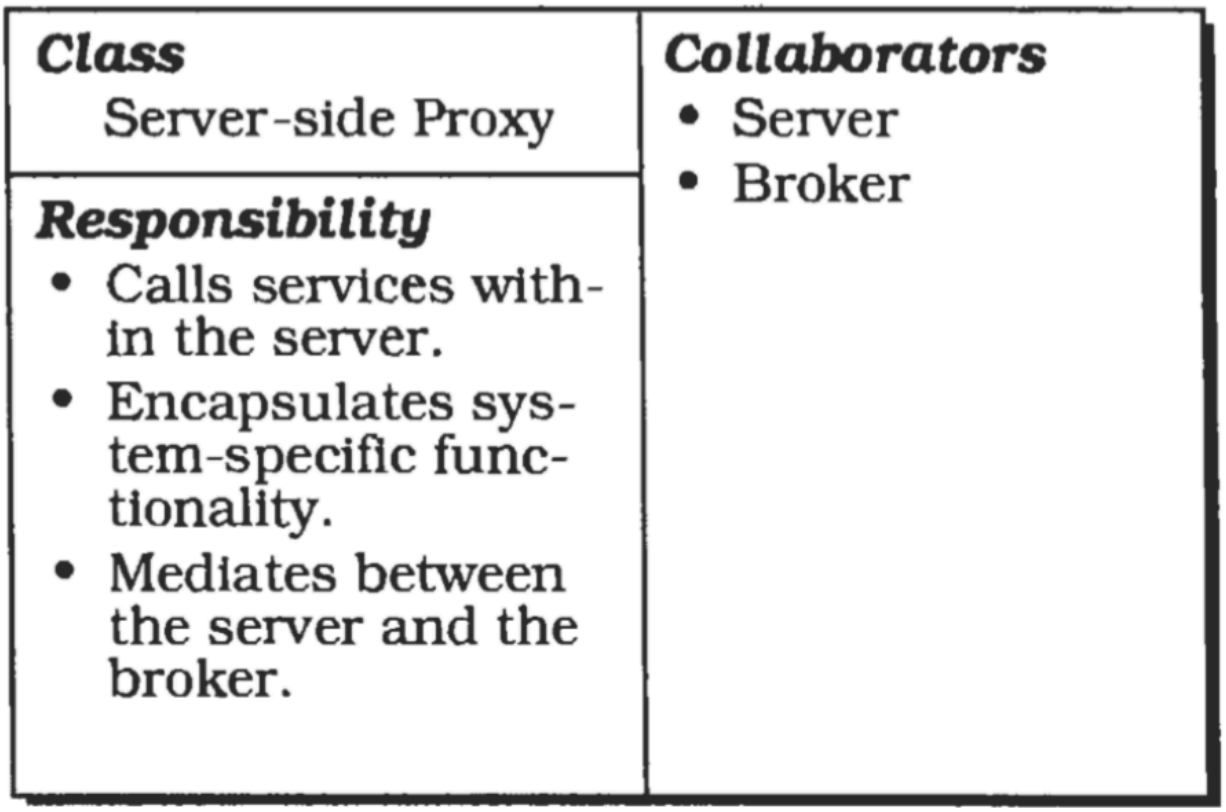
- **Client-side Proxies**: Handle requests from the client side.
- **Server-side Proxies**: Handle requests from the server side.

Diagram: CRC Diagram for Proxies

- **Client-side Proxies**

Class	***Collaborators***
Client-side Proxy	• Client • Broker
Responsibility	
• Encapsulates system-specific functionality. • Mediates between the client and the broker.	

- **Server-side Proxies**



---

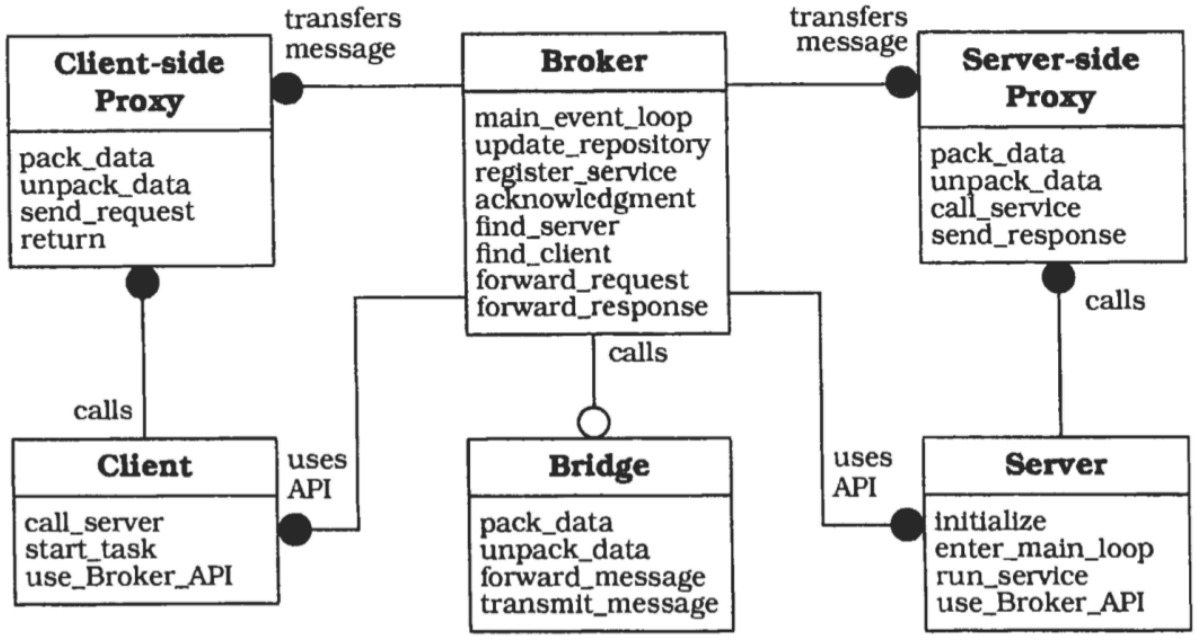
5. Bridges

Bridges manage communication between brokers in heterogeneous networks, hiding differences in operating systems or network protocols.

Diagram: CRC Diagram for Bridges

Class Bridge	Collaborators • Broker • Bridge
Responsibility • Encapsulates network-specific functionality. • Mediates between the local broker and the bridge of a remote broker.	

Broker Pattern



Implementation Steps

1. Define the object model for system communication.
2. Decide the kind of component interoperability the system should offer.
3. Design APIs that the broker component will provide.
4. Use proxy objects to hide implementation details from clients and servers.
5. Design the broker and proxy components in parallel.
6. Develop compilers for Interface Definition Language (IDL).

Variants of Broker Pattern

- **Direct Communication Broker System**
- **Message Passing Broker System**
- **Trader System**
- **Adapter Broker System**
- **Callback Broker System**

Known Usages

- **CORBA**: Common Object Request Broker Architecture for enabling communication between systems in different languages.
- **Microsoft OLE**: Object Linking and Embedding for communication between different applications.
- **World Wide Web**: Internet communication through HTTP and web servers.

Consequences of Using the Broker Pattern

Benefits:

- **Location Transparency**: Clients don't need to know where the server is located.
- **Extensibility**: New services can be added or changed dynamically.
- **Portability**: The system can operate on different platforms.
- **Interoperability**: Different systems can communicate using the broker.

Liabilities:

- **Reduced Efficiency**: Adding a broker layer may add overhead.
- **Fault Tolerance**: The system relies on the broker's stability.
- **Testing and Debugging**: More complex due to distributed nature.