Module 6 CS07

Patterns – Part 1 Suppl 3

Distributed Systems Brokers/Bridge

Harvinder S Jabbal
SSZG653 Software Architectures

BITS Pilani
Pilani|Dubai|Goa|Hyderabad

Categories

From Mud to Structure.

- Patterns in this category help you to avoid a 'sea' of components or objects.
- In particular, they support a controlled decomposition of an overall system task into cooperating subtasks.
- The category includes
  - the Layers pattern
  - the Pipes and Filters pattern
  - the Blackboard pattern

Distributed Systems.

- This category includes one pattern.
  - Broker
- and refers to two patterns in other categories,
  - Microkernel
  - Pipes and Filters
- The Broker pattern provides a complete infrastructure for distributed applications.
- The Microkernel and Pipes and Filters patterns only consider distribution as a secondary concern and are therefore listed under their respective primary categories.

Interactive Systems.

- This category comprises two patterns,
  - the Model-View-Controller pattern (well-known from Smalltalk,)
  - the Presentation-Abstraction-Control pattern.
- Both patterns support the structuring of software systems that feature human-computer interaction.

Adaptable Systems.

- This category includes
  - The Reflection pattern
  - the Microkernel pattern
- strongly support extension of applications and their adaptation to evolving technology and changing functional requirements.

Distributed Systems

innovate achieve lead

- Broker
  - The Microkernel and Pipes and Filters patterns only consider distribution as a secondary concern and are therefore listed under their respective primary categories.

innovate achieve lead

BITS Pilani

Pilani|Dubai|Goa|Hyderabad

Distributed Systems: Broker

Broker

- The Broker architectural pattern can be used to
  - structure distributed software systems
  - with decoupled components
  - that interact by remote service invocations.
- A broker component is responsible
  - for coordinating communication,
  - such as forwarding requests,
  - as well as for transmitting results
  - and exceptions.

Context

- Your environment is a
  - distributed and
  - possibly heterogeneous system with
    - independent
    - cooperating components

Problem

- Building a complex software system as a set of decoupled and inter-operating components, rather than as a monolithic application, results in greater flexibility, maintainability and changeability.
- By partitioning functionality into independent components the system becomes potentially distributable and scalable.
- However, when distributed components communicate with each other, some means of inter-process communication is required.
- If components handle communication themselves, the resulting system faces several dependencies and limitations.

Example

- The system becomes dependent on the communication mechanism used, clients need to know the location of servers, and in many cases the solution is limited to only one programming language.
- Services for adding, removing, exchanging, activating and locating components are also needed.
- Applications that use these services should not depend on system-specific details to guarantee portability and interoperability, even within a heterogeneous network.

developer's viewpoint

- There should essentially be no difference between developing software for centralized systems and developing for distributed ones.
- An application that uses an object should only see the interface offered by the object.
- It should not need to know anything about the implementation details of an object, or about its physical location.

Forces balanced by Broker Architecture

innovate achieve lead

- Components should be able to access services provided by others through remote, location-transparent service invocations.
- You need to exchange, add, or remove components at run-time.
- The architecture should hide system and implementation-specific details from the users of components and services.

Solution

- Introduce a broker component to achieve better decoupling of clients and servers.
- Servers register themselves with the broker, and make their services available to clients through method interfaces.
- Clients access the functionality of servers by sending requests via the broker.
- A broker's tasks include locating the appropriate server, forwarding the request to the server and transmitting results and exceptions back to the client.

How an Application accesses distributed services

- By using the Broker pattern, an application can access distributed services simply by sending message calls to the appropriate object, instead of focusing on low-level inter-process communication.
- In addition, the Broker architecture is flexible, in that it allows dynamic change, addition, deletion, and relocation of objects.

How it works

- The Broker pattern reduces the complexity involved in developing distributed applications, because it makes distribution transparent to the developer.
- It achieves this goal by introducing an object model in which distributed services are encapsulated within objects.
- Broker systems therefore offer a path to the integration of two core technologies:
    - distribution and object technology.
- They also extend object models from single applications to distributed applications consisting of decoupled components that can run on heterogeneous machines and that can be written in different programming languages.

Structure

- The Broker architectural pattern comprises six types of participating components:
  - Clients,
  - servers,
  - brokers,
  - bridges,
  - client-side proxies and
  - server-side proxies.

Server

- A Server implements objects that expose their functionality through interfaces that consist of operations and attributes.
- These interfaces are made available either through an interface definition language (IDL) or through a binary standard.
- Interfaces typically group semantically-related functionality.
- There are two kinds of servers:
    - Servers offering common services to many application domains.
    - Servers implementing specific functionality for a single application domain or task.

CRC diagram

Class Server	**Collaborators**
Responsibility • Implements services. • Registers itself with the local broker. • Sends responses and exceptions back to the client through a server-side proxy.	• Server-side Proxy • Broker

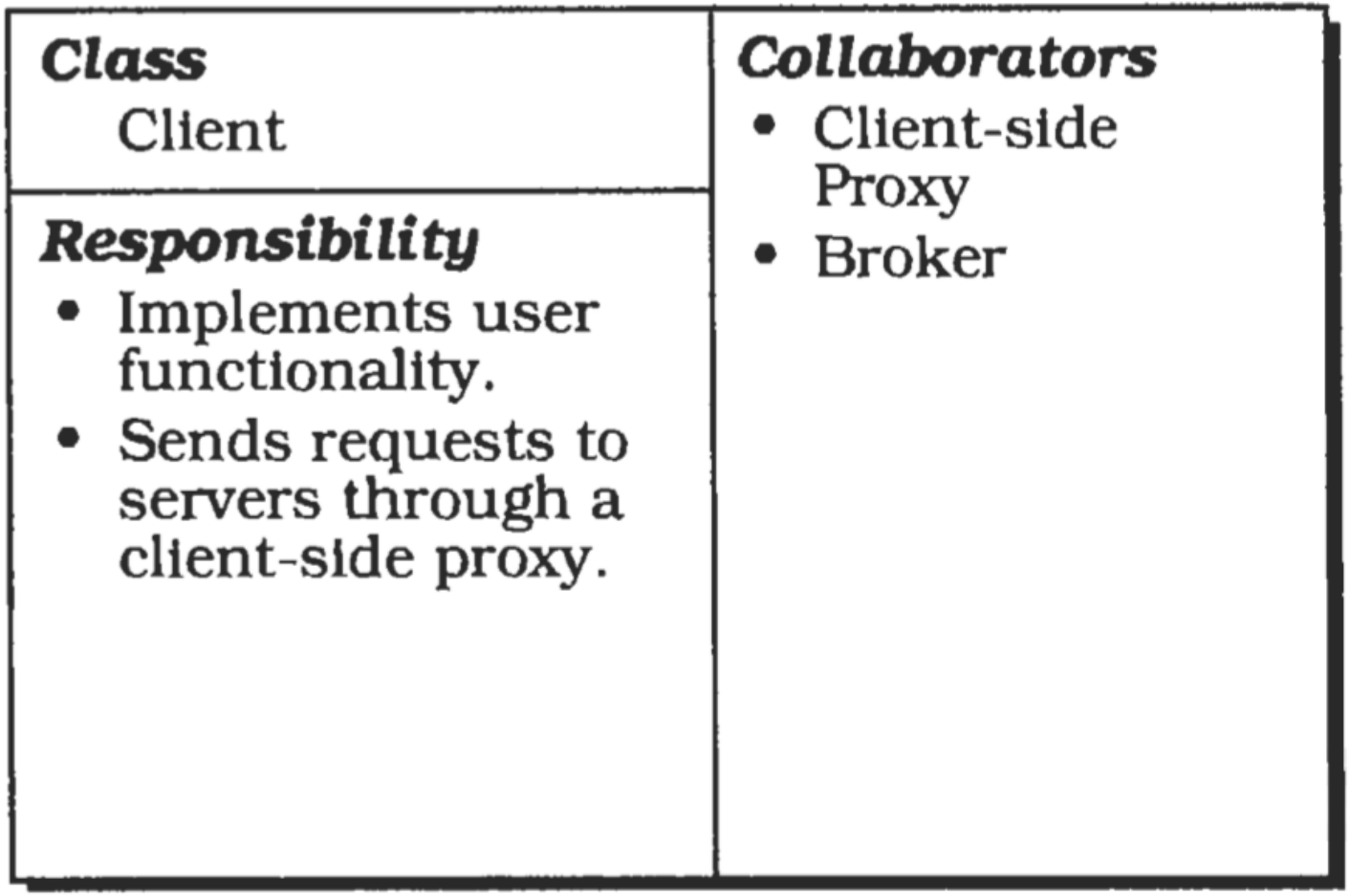
Server: Illustration

- WWW servers that provide access to HTML (Hypertext Markup Language) pages.
- WWW servers are implemented as http daemon processes (hypertext transfer protocol daemon) that wait on specific ports for incoming requests.
- When a request arrives at the server, the requested document and any additional data is sent to the client using data streams.
- The HTML pages contain documents as well as CGI (Common Gateway interface) scripts for remotely-executed operations on the network host-the remote machine from which the client received the HTML- page.
- A CGI script may be used to allow the user fill out a form and submit a query, for example a search request for vacant hotel rooms.
- To display animations on the client's browser, Java 'applets' are integrated into the HTML documents.
- For example, one of these Java applets animates the route between one place and another on a city map.
- Java applets run on top of a virtual machine that is part of the WWW browser.
- CGI scripts and Java applets differ from each other: CGI scripts are executed on the server machine, whereas Java applets are transferred to the WWW browser and then executed on the client machine.

Client

- Clients are applications that access the services of at least one server.
- To call remote services, clients forward requests to the broker.
- After an operation has executed they receive responses or exceptions from the broker.
- The interaction between clients and servers is based on a dynamic model, which means that servers may also act as clients.
- This dynamic interaction model differs from the traditional notion of Client-Server computing in that the roles of clients and servers are not statically defined.
- From the viewpoint of an implementation, you can consider clients as applications and servers as libraries-though other implementations are possible.
- Note that clients do not need to know the location of the servers they access.
- This is important, because it allows the addition of new services and the movement of existing services to other locations, even while the system is running.

CRC Diagram


Class
Client
Responsibility
• Implements user functionality.
• Sends requests to servers through a client-side proxy.
Collaborators
• Client-side Proxy
• Broker

Client: Illustration

- In the context of the Broker pattern, the clients are the available WWW browsers.
- They are not directly connected to the network. Instead, they rely on Internet providers that offer gateways to the Internet, such as vsnl.
- WWW browsers connect to these workstations, using either a modem or a leased line.
- When connected they are able to retrieve data streams from httpd servers, interpret this data and initiate actions such as the display of documents on the screen or the execution of Java applets.

Broker

- A broker is a messenger that is responsible for the transmission of requests from clients to servers, as well as the transmission of responses and exceptions back to the client.
- A broker must have some means of locating the receiver of a request based on its unique system identifier. A broker offers APIs (Application Programming Interfaces) to clients and servers that include operations for registering servers and for invoking server methods.
- When a request arrives for a server that is maintained by the local broker, the broker passes the request directly to the server.
- If the server is currently inactive, the broker activates it.
- All responses and exceptions from a service execution are forwarded by the broker to the client that sent the request.
- If the specified server is hosted by another broker, the local broker finds a route to the remote broker and forwards the request using this route.
- There is therefore a need for brokers to interoperate.
- Depending on the requirements of the whole system, additional services-such as name services or marshalling services may be integrated into the broker.

CRC Diagram

Class Broker	***Collaborators***
Responsibility • (Un-)registers servers. • Offers APIs. • Transfers messages. • Error recovery. • Interoperates with other brokers through bridges. • Locates servers.	• Client • Server • Client-side Proxy • Server-side Proxy • Bridge

Some Definitions

- In this pattern description we distinguish between local and remote brokers.
  - A local broker is running on the machine currently under consideration.
  - A remote broker is running on a remote network node.
- Name services provide associations between names and objects.
  - To resolve a name, a name service determines which server is associated with a given name.
  - In the context of Broker systems, names are only meaningful relative to a name space.
- Marshalling is the semantic-invariant conversion of data into a machine independent format such as
  - ASN.1 (Abstract Syntax Notation or
  - ONC XDR [external Data Representation).
- Marshalling performs the reverse transformation.

Broker: Illustration

- A broker may be a combination of an Internet gateway and the Internet infrastructure itself.
- Every information exchange between a client and a server must pass through the broker.
- A client specifies the information it wants using unique identifiers called URLs (Universal Resource Locators).
- By using these identifiers the broker is able to locate the required services and to route the requests to the appropriate server machines.
- When a new server machine is added, it must be registered with the broker.
- Clients and servers use the gateway of their Internet provider as an interface to the broker.

Client-side proxies

- Client-side proxies represent a layer between clients and the broker.
- This additional layer provides transparency, in that a remote object appears to the client as a local one.
- In detail, the proxies allow the hiding of implementation details from the clients such as:
    - The inter-process communication mechanism used for message transfers between clients and brokers.
    - The creation and deletion of memory blocks.
    - The marshalling of parameters and results.
- In many cases, client-side proxies translate the object model specified as part of the Broker architectural pattern to the object model of the programming language used to implement the client.

CRC Diagram

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

Server Side Proxy

- Sewer-side proxies are generally analogous to Client-side proxies.
- The difference is that they are responsible for receiving requests, unpacking incoming messages, marshalling the parameters, and calling the appropriate service.
- They are used in addition for marshalling results and exceptions before sending them to the client.

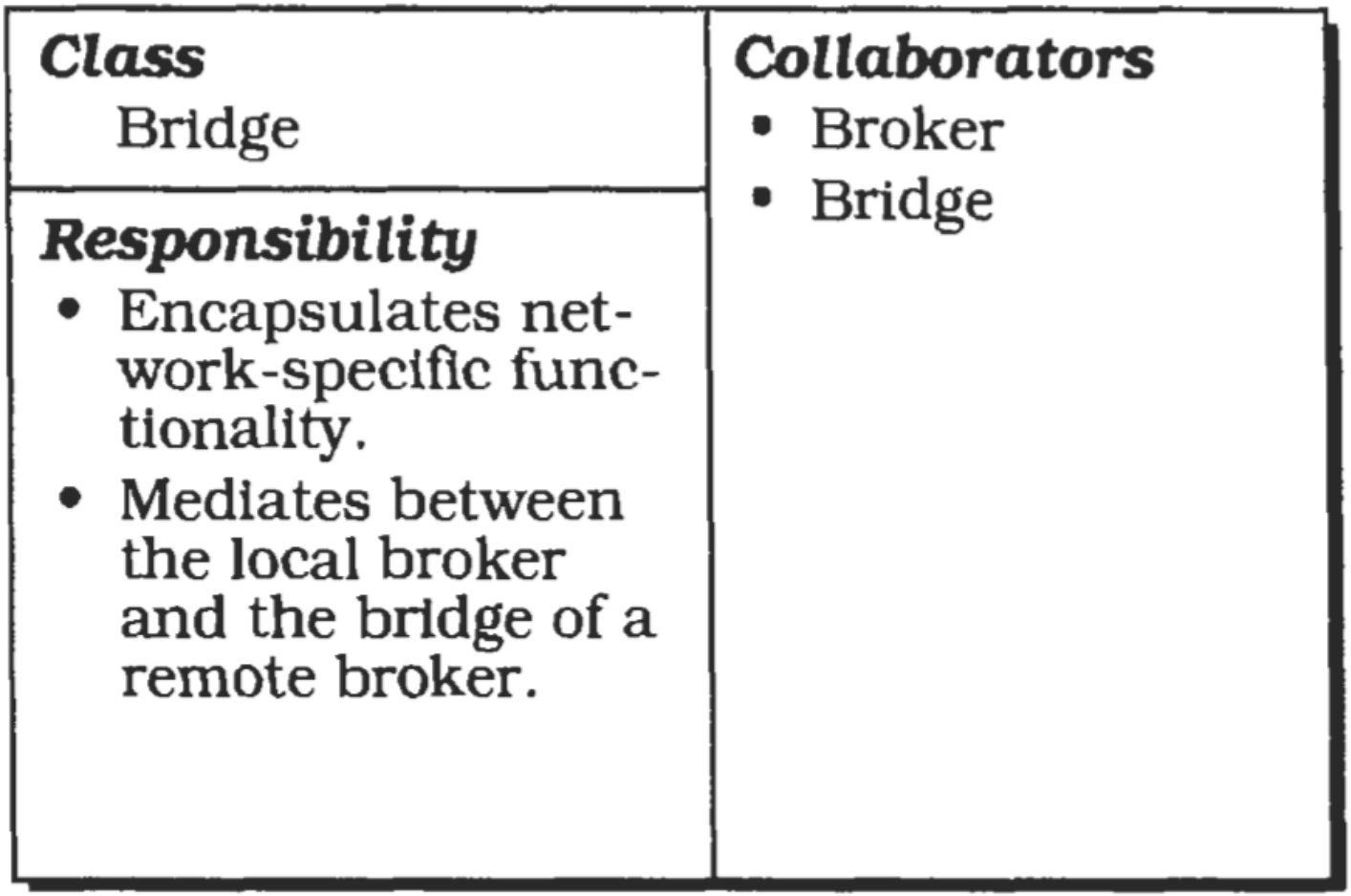
CRC Diagram

Class	Collaborators
Server-side Proxy	• Server • Broker
Responsibility	
• Calls services within the server. • Encapsulates system-specific functionality. • Mediates between the server and the broker.	

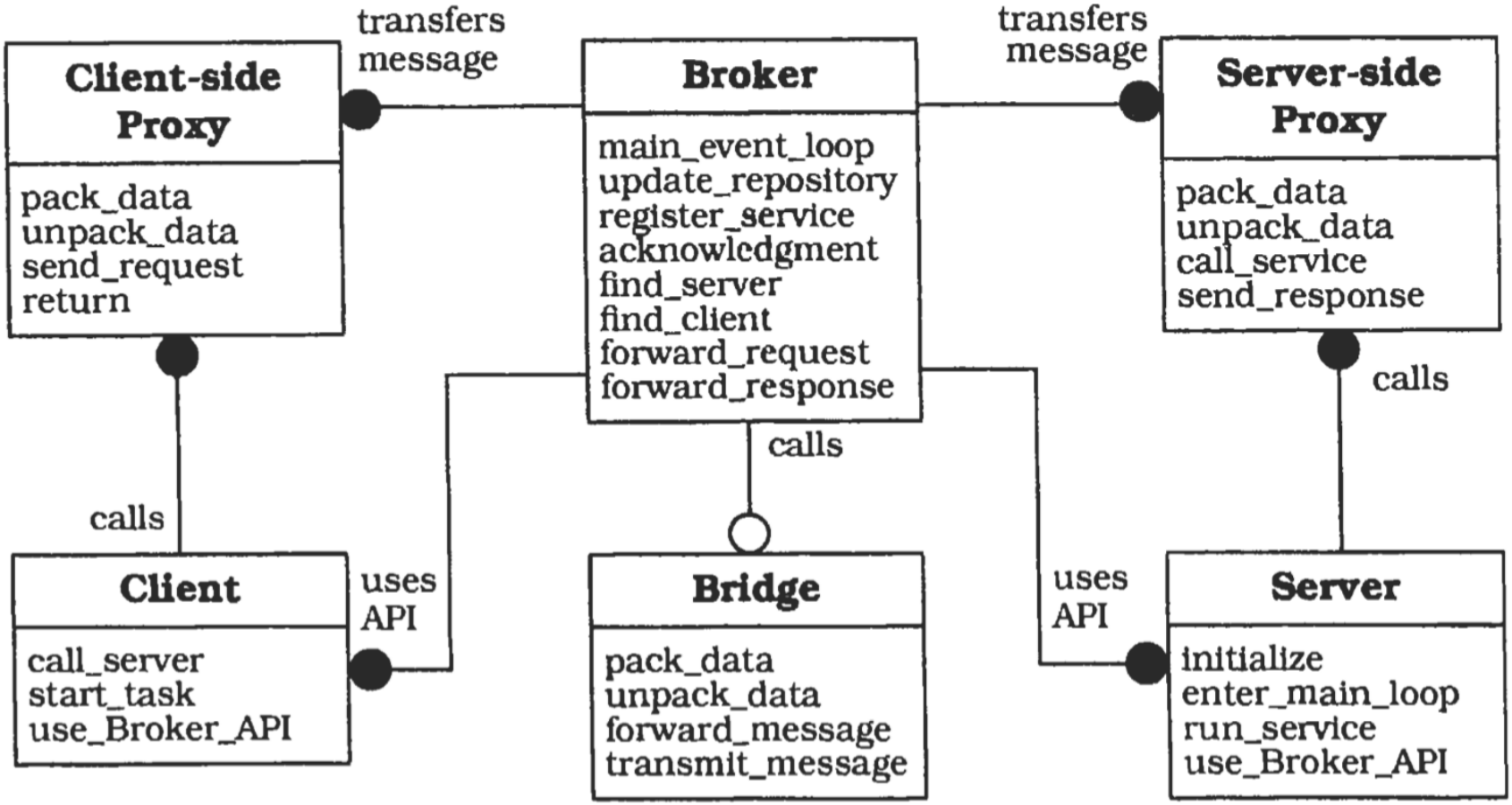
Bridges

- Bridges are optional components used for hiding implementation details when two brokers interoperate.
- Suppose a Broker system runs on a heterogeneous network.
- If requests are transmitted over the network, different brokers have to communicate independently of the different network and operating systems in use.
- A bridge builds a layer that encapsulates all these system-specific details.

CRC Diagram


Class
Bridge
Responsibility
• Encapsulates net-work-specific func-tionality.
• Mediates between the local broker and the bridge of a remote broker.
Collaborators
• Broker
• Bridge

Broker System



Implementation

1. Define an object model, or use an exlsling model
2. Decide which kind of component-interoperability the system should offer.
3. Specify the APIs the broker component provides for collaborating with clients and servers.
4. Use proxy objects to hide implementation details from clients and servers.
5. Design the broker component in parallel with steps 3 and 4. (Note the steps involved)
6. Develop IDL compilers.

Variants

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

Known Usages

- CORBA
- IBM SOM/DSOM
- Microsoft's OLE 2.x
- World Wide Web
- ATM-P.

Consequences: benefits

- Location Transparency
- Changeability and extensibility of components
- Portability of a Broker system
- Interoperability between dzflerent Broker systems.
- Reusability
- Testing and Debugging

Consequences: Liabilities

- Restricted efficiency
- Lower fault tolerance
- Testing and Debugging