DECSAI
Departamento de Ciencias de la Computación e I.A.
Universidad de Granada

Middleware – Broker

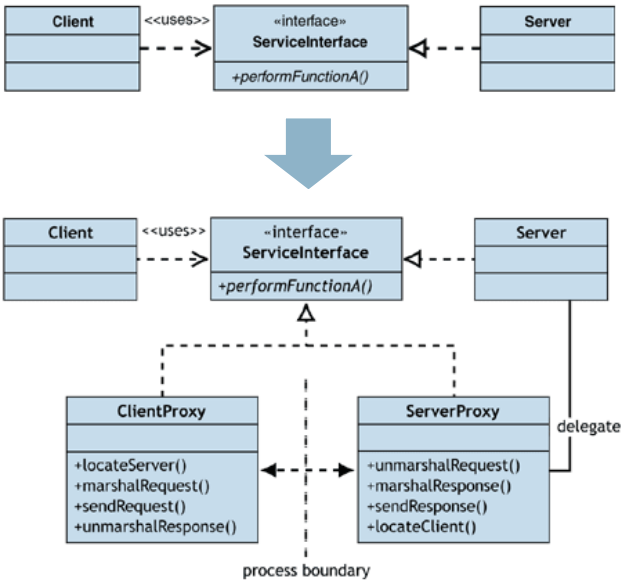
© Fernando Berzal, berzal@acm.org

Middleware – Broker

- Proxies
- El patrón de diseño BROKER
  - Servicio de nombres vs. Intermediario
  - Escenarios de uso.
  - Diseño, implementación y pruebas.
  - Beneficios y limitaciones
  - Variantes
- CORBA [Common Object Request Broker Architecture]
  - Llamada a un método en CORBA
  - Diseño de un ORB [Object Request Broker]

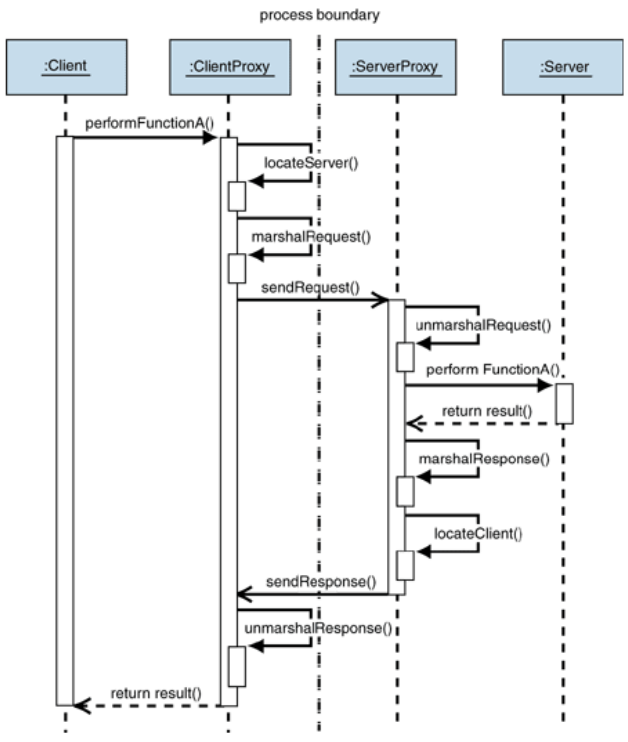
Proxies

Distribución



Proxies

Distribución



Broker

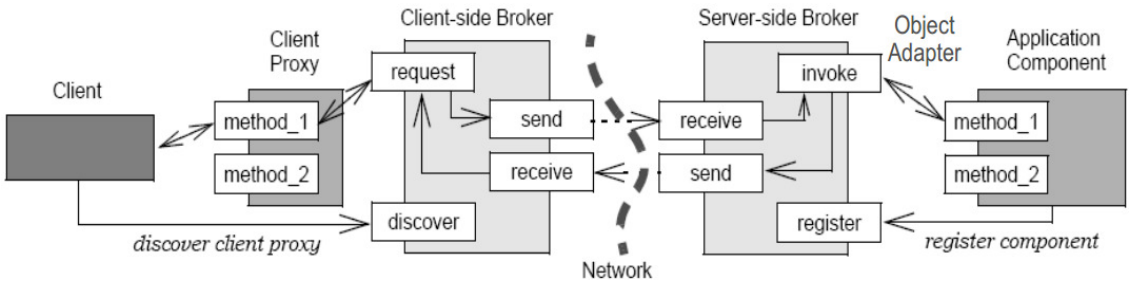
El patrón de diseño Broker se utiliza para organizar sistemas distribuidos con componentes desacoplados que interaccionan realizando invocaciones remotas a servicios.

El **broker** es el responsable de coordinar la comunicación (reenviar solicitudes & transmitir resultados o excepciones).

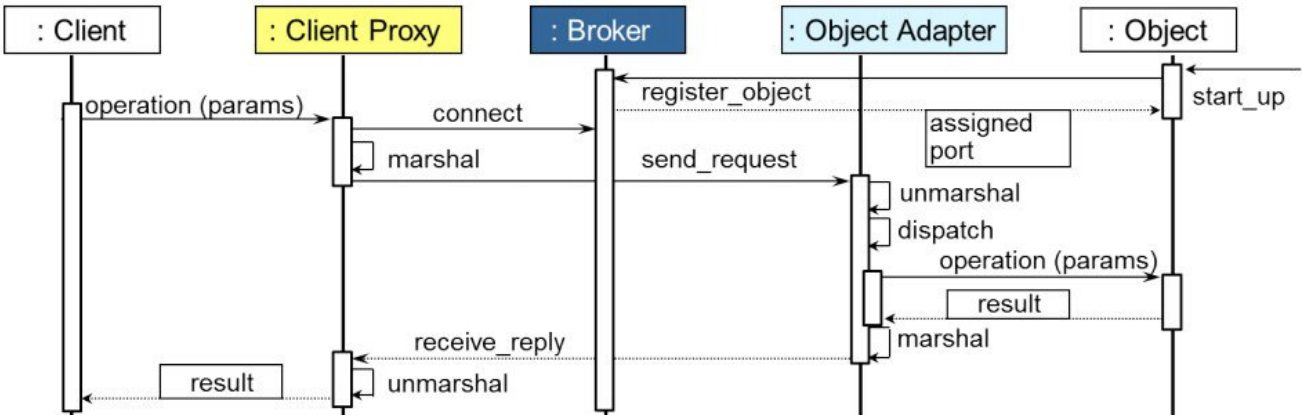
EJEMPLOS

- Microsoft OLE
  [Object Linking and Embedding]
- OMG CORBA
  [Common Object Request Broker Architecture]

Broker

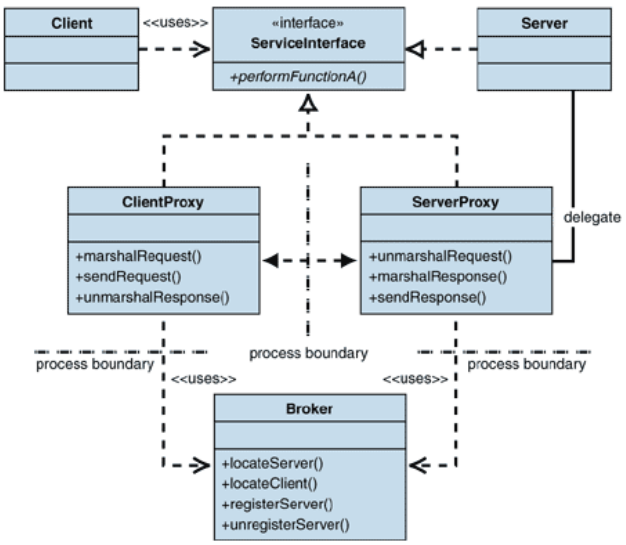


Broker



Broker

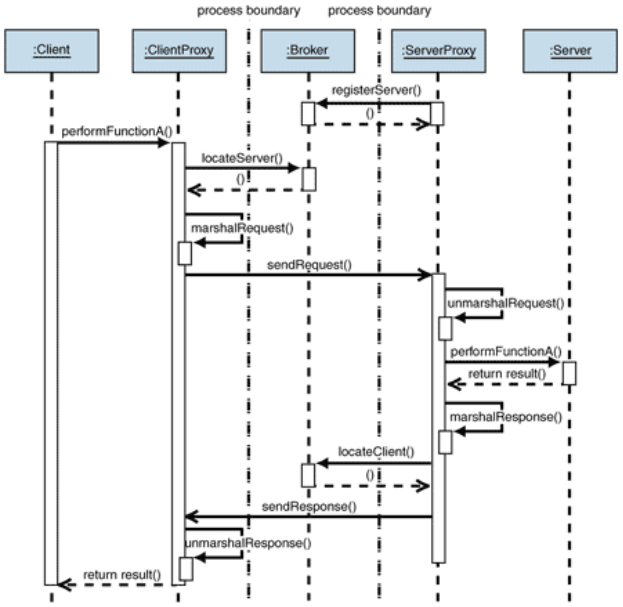
Broker como servicio de nombres



Microsoft®
patterns & practices
proven practices for predictable results

Broker

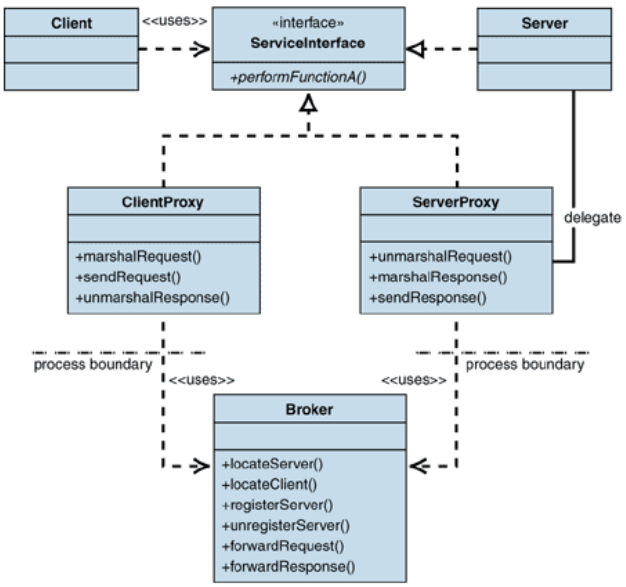
Broker como servicio de nombres



Microsoft®
patterns & practices
proven practices for predictable results

Broker

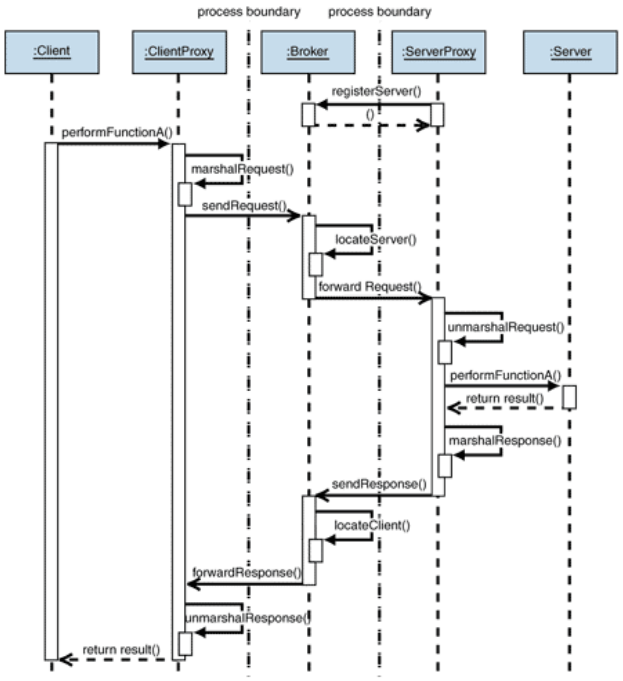
Broker como intermediario



Microsoft®
patterns & practices
proven practices for predictable results

Broker

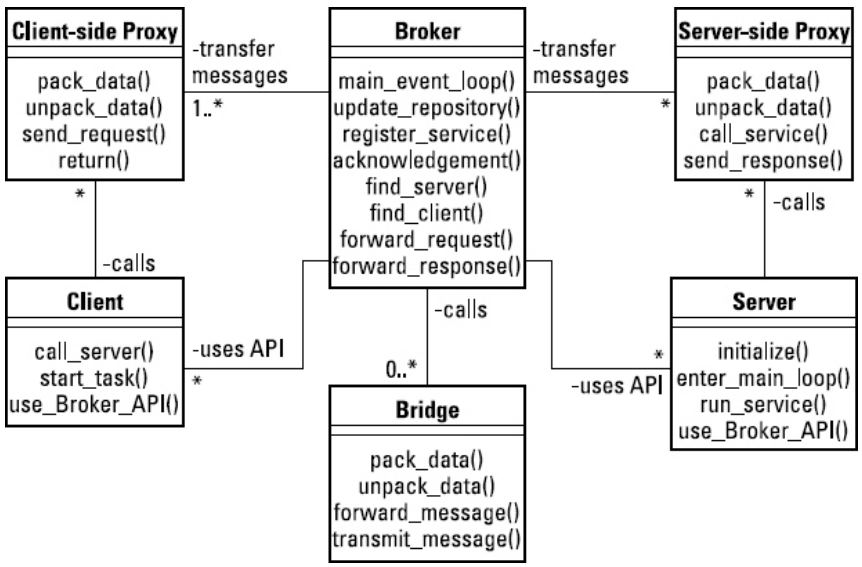
Broker como intermediario



Microsoft®
patterns & practices
proven practices for predictable results

Broker

Diagrama de clases completo



Broker

Class: Broker	Collaborators
Responsibilities • Register and unregister servers • Provide APIs • Transfer messages • Error recovery • Interoperate with other broker systems through bridges • Locate servers	• Client • Server • Client-side proxy • Server-side proxy • Bridge

Class: Client	Collaborators
Responsibilities • Implement user functionality • Send request to servers through client-side proxies	• Client-side proxy • Broker

Class: Server	Collaborators
Responsibilities • Implement services • Register itself with local broker • Send responses and exceptions to client through server-side proxy	• Server-side proxy • Broker

Broker

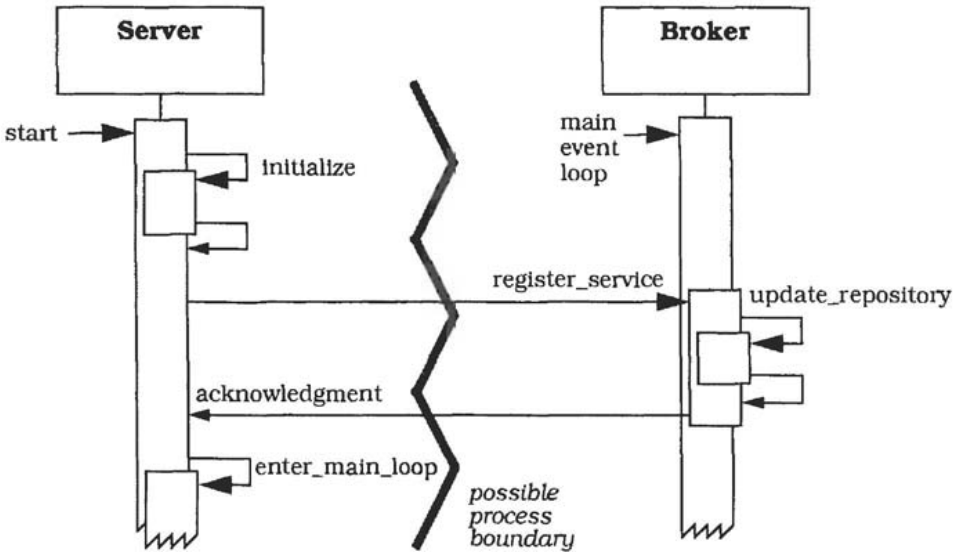
Class: Client-side proxy	Collaborators
Responsibilities • Encapsulate system-specific functionality • Mediate between client and broker	• Client • Broker

Class: Server-side proxy	Collaborators
Responsibilities • Call services within the server • Encapsulate system-specific functionality • Mediate between the server and the broker	• Server • Broker

Class: Bridge	Collaborators
Responsibilities • Encapsulate network-specific functionality • Mediate between the local broker and the bridge of a remote broker	• Broker • Bridge

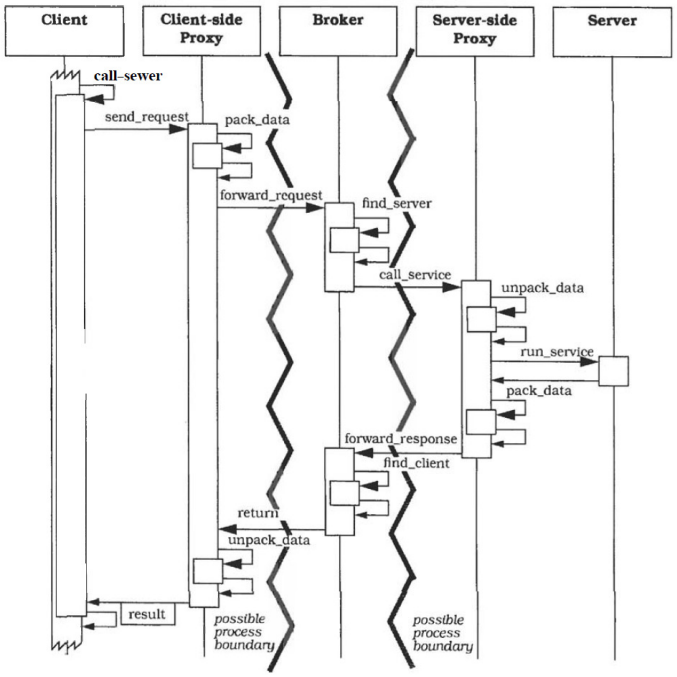
Broker

ESCENARIO I: Registro de un servicio



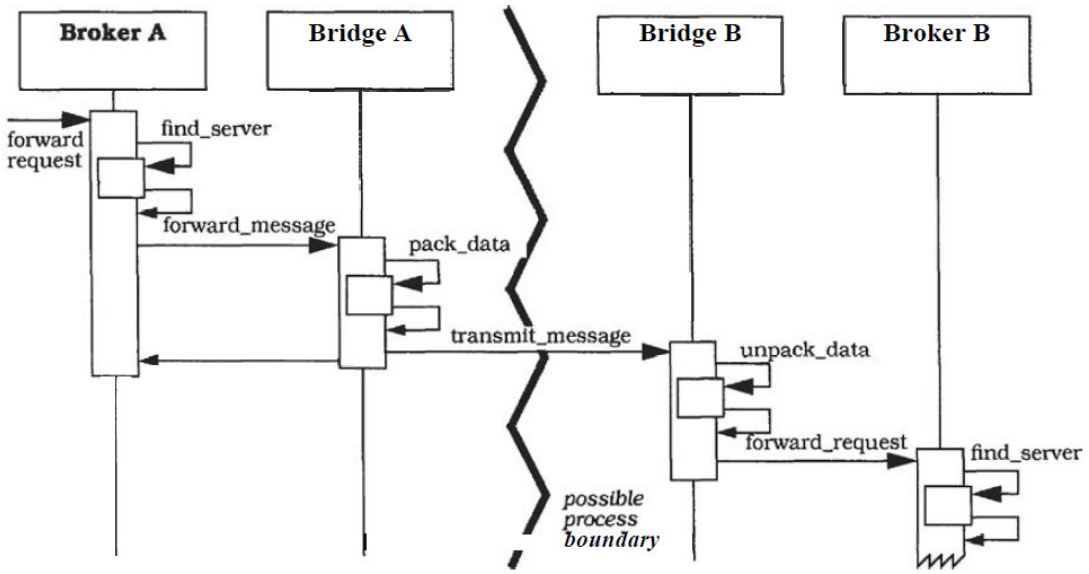
Broker

ESCENARIO II: Invocación de un servicio local



Broker

ESCENARIO III: Interacción entre brokers vía puentes



Broker

- Útil para sistemas de información complejos, que evolucionan y crecen con el tiempo: el broker se encarga de que podamos acceder a los distintos servicios del sistema sin necesidad de conocer su localización.
- Las arquitecturas basadas en el uso de brokers nos permiten replicar y migrar servicios.
- Si la arquitectura define los protocolos de comunicación utilizados, se pueden construir sistemas distribuidos heterogéneos con componentes independientes que cooperan entre sí.

Broker

Requisitos

- Mecanismo de comunicación entre procesos [IPC: Inter-Process Communication].
- Servicios para añadir, eliminar, intercambiar, activar y localizar componentes (que deberían ser independientes de los detalles específicos del sistema para garantizar su portabilidad y interoperabilidad en entornos heterogéneos).

Broker

Implementación

1. Definir el **modelo de objetos** (diseño OO de la interacción entre clientes y servidores).
2. Determinar el **nivel de interoperabilidad**:
   - API, a nivel del código fuente (mayor portabilidad).
   - ABI, a nivel binario (mayor rendimiento).
3. Diseñar el **API/ABI** ofrecido por el broker.
4. Ocultar detalles de implementación mediante **proxies** (tanto para clientes como para servidores).
5. Diseñar el **broker** en sí…
6. Desarrollar compiladores de **IDL** [Interface Definition Language].

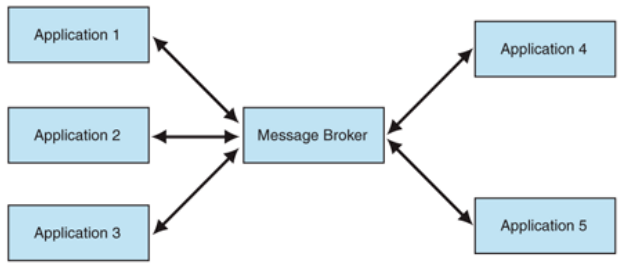
Broker

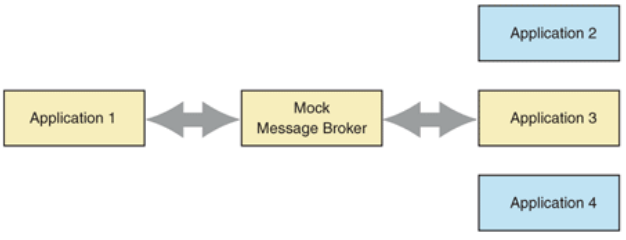
Diseño del broker

- Protocolo de interacción con los proxies (C & S).
- Puentes (routing de mensajes a otros brokers).
- Protocolo de intercambio de solicitudes y respuestas.
- Marshalling/unmarshalling (en el broker o en los proxies).
- Buffers de mensajes para comunicación asíncrona.
- Directorio de servicios (IDs → direcciones físicas).
- Invocación de métodos estática (en tiempo de compilación) vs. dinámica (en tiempo de ejecución, requiere el uso de registros de servidores).
- Gestión de errores (tanto en los servicios a los que se accede como en las comunicaciones).

Broker

Pruebas





Broker

Ventajas (frente a una arquitectura monolítica)

- Mayor flexibilidad, mantenibilidad y adaptabilidad.
- Mayor escalabilidad (posibilidad de distribución).
- Servidores invisibles a los clientes (p.ej. migración).
- Portabilidad (API/ABI del middleware independiente del sistema operativo o de la red de comunicación).
- Interoperabilidad entre brokers (facilita la construcción de sistemas distribuidos heterogéneos).
- Componentes reutilizables (interfaces públicas)

Broker

Posibles inconvenientes/limitaciones

- El broker se convierte en un punto crítico de la arquitectura (si falla, el sistema se viene abajo).
- Rendimiento (al ocultar detalles de implementación, algunas optimizaciones no se pueden aprovechar).
- Si los componentes manejan directamente la comunicación, el sistema puede ser dependiente del mecanismo de comunicación utilizado, limitado a una única plataforma de programación y perder la transparencia con respecto a la localización de los servicios.
- La realización de labores de prueba y depuración puede ser más compleja.

Broker

Variantes

COMUNICACIÓN DIRECTA [DIRECT COMMUNICATION BROKER SYSTEM]

- Comunicación directa entre cliente y servidor:
  El broker se emplea para establecer la conexión entre cliente y servidor. Una vez establecida la conexión, el broker no vuelve a intervenir.

- Resuelve algunos problemas de eficiencia asociados al uso de brokers.

Broker

Variantes

TRADER [BROKER SYSTEM]

- La solicitud del cliente no se envía a un servidor particular, sino a un servicio (no solicita el identificador de un servidor, sino el de un servicio).

- El broker controla qué servidores ofrecen qué servicios (e independiza al cliente del proveedor del servicio).

Broker

Variantes

ADAPTER [BROKER SYSTEM]

- La interfaz del servidor con el broker se oculta con una capa adicional controlada por el propio broker: el adaptador [adapter].
- Si clientes y servidores se encuentran en el mismo sistema que el broker, el adaptador permite la comunicación directa entre clientes y servidores (p.ej. usando memoria compartida, mucho más rápida que un mecanismo de comunicación entre procesos).

Broker

Variantes

CALLBACK [BROKER SYSTEM]

- Implementación de sistemas reactivos (dirigidos y controlados por eventos, a los que se reacciona).
- El broker no distingue entre clientes y servidores: cuando sucede un evento, el broker invoca el método "callback" de cualquier componente que se haya registrado para recibir notificaciones del evento.
- Más sobre esto cuando lleguemos a los sistemas de tipo **publish/subscribe**…

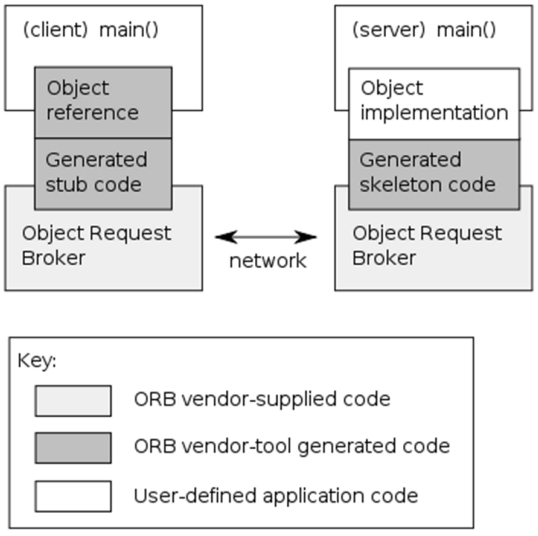
Broker

Variantes

MESSAGE PASSING [BROKER SYSTEM]

- Cada mensaje incluye un identificador que el servidor utiliza para determinar qué acción realizar.
- Los mensaje contienen metadatos además de los datos [payload]: tipo de mensaje, estructura y otros atributos que pueda necesitar el servidor.
- Usado en sistemas en los que se transmiten principalmente datos, más que solicitudes de servicio.

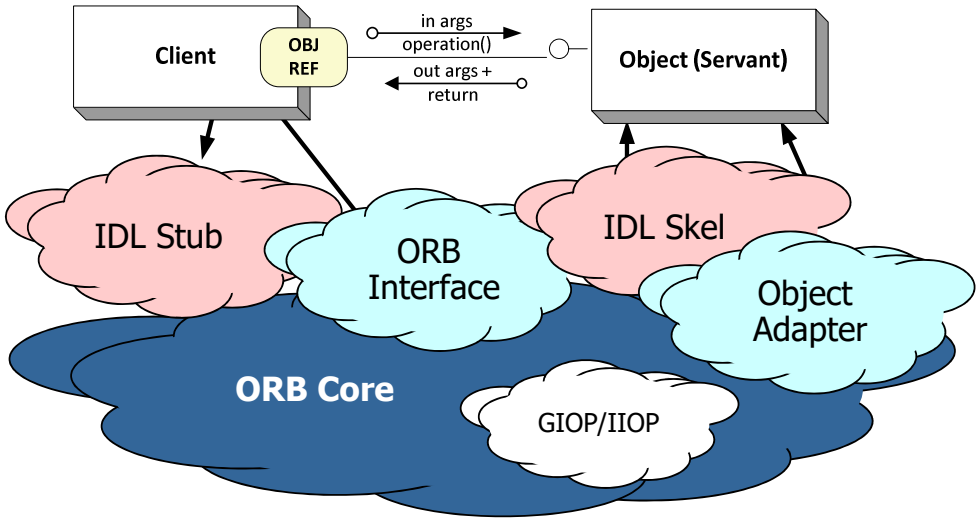
CORBA



Common Object Request Broker Architecture

https://en.wikipedia.org/wiki/Common_Object_Request_Broker_Architecture

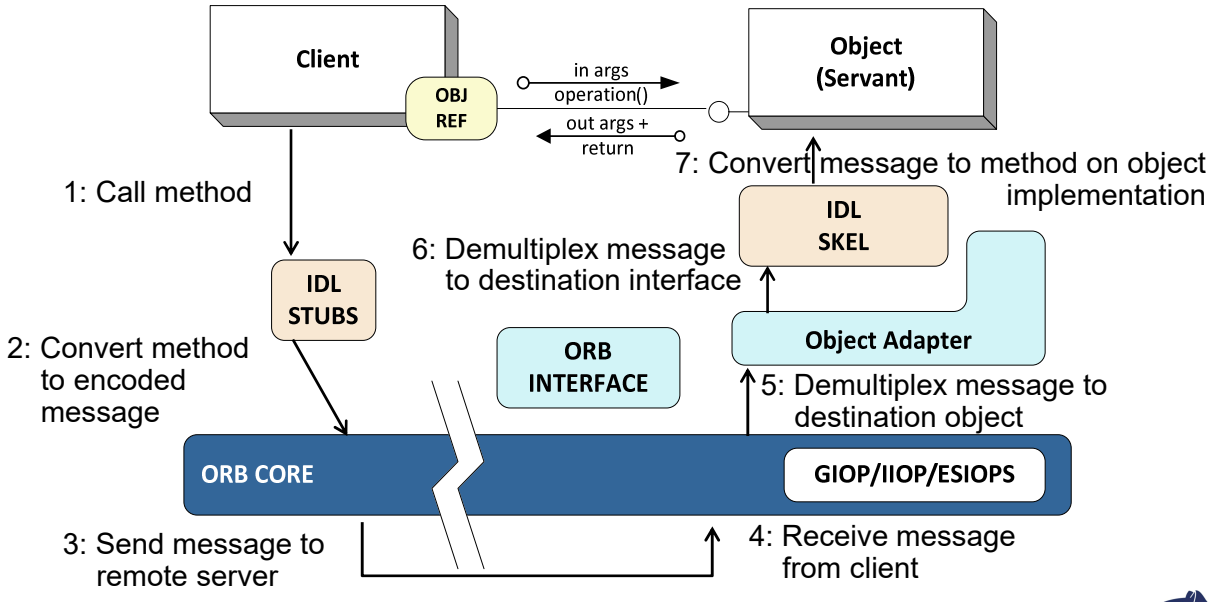
CORBA



ORB [Object Request Broker]

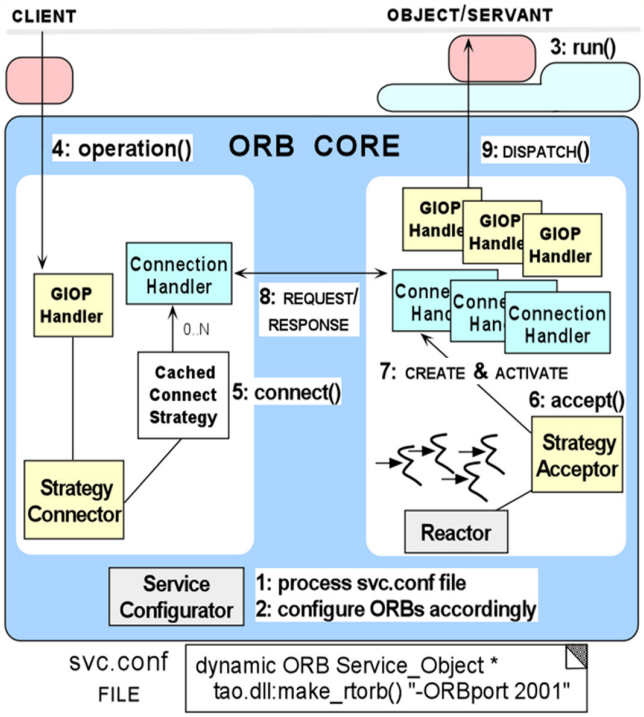
http://www.dre.vanderbilt.edu/~schmidt/PDF/ORB-patterns.pdf

CORBA



Llamada a un método en CORBA

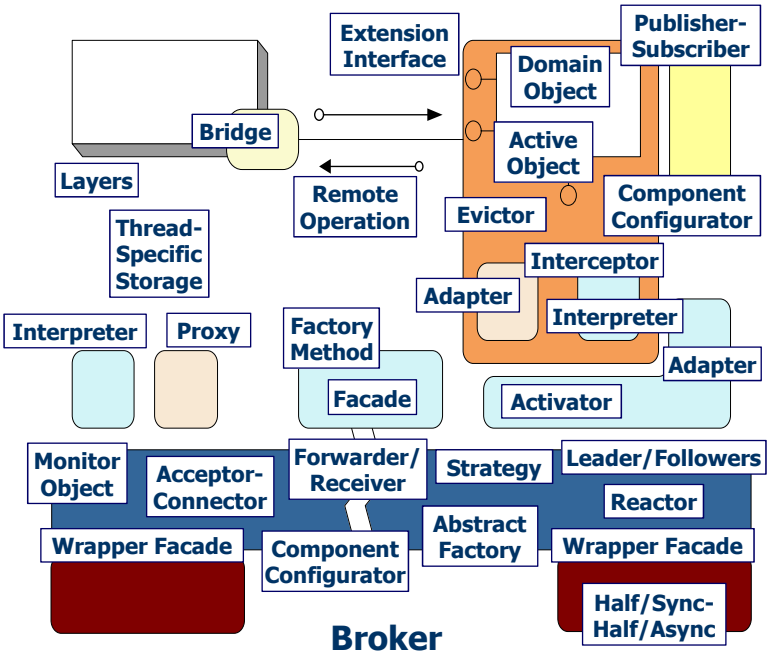
CORBA



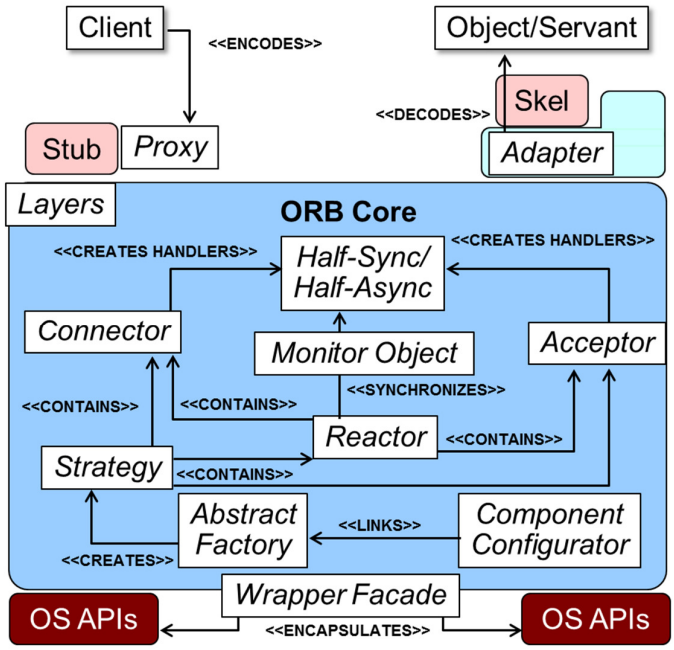
Llamada a un método en CORBA, e.g. TAO

CORBA

Diseño de un ORB



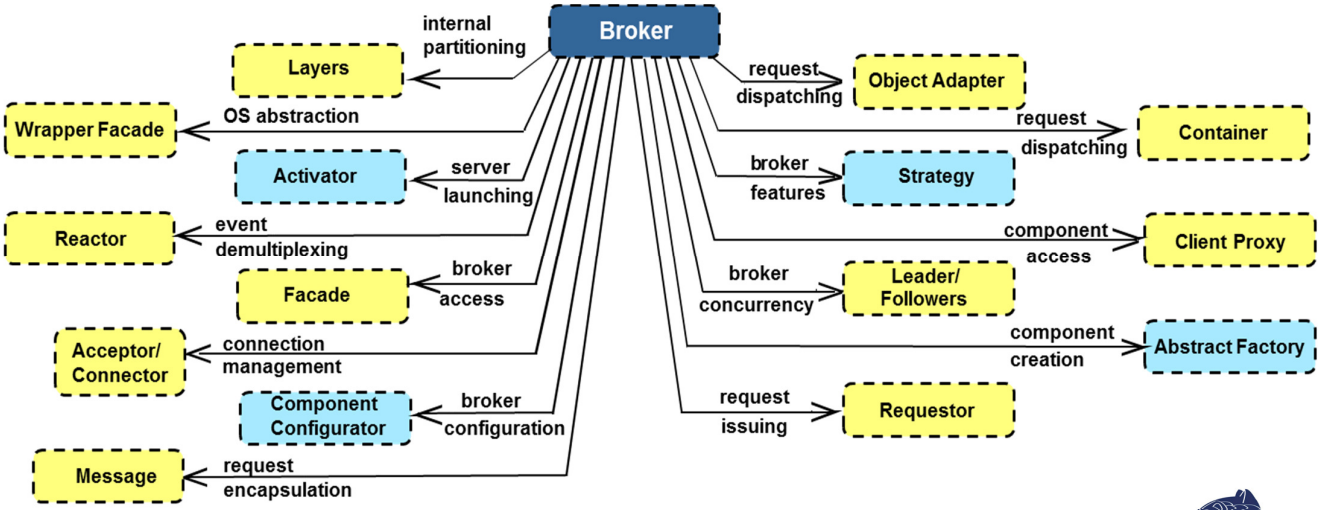
Diseño de un ORB



Pattern	Problem
Layers	Structuring broker internal design to enable reuse & clean separation of concerns
Wrapper Façade	Encapsulating low-level system functions to enhance portability
Reactor	Demuxing broker core events efficiently
Acceptor-Connector	Managing broker connections efficiently
Half-Sync/Half-Async	Enhancing broker scalability by processing requests concurrently
Monitor Object	Efficiently synchronize the Half-Sync/Async request queue
Strategy	Interchanging internal broker mechanisms transparently
Abstract Factory	Consolidating broker mechanisms [i.e. strategies] into groups of semantically compatible strategies
Component Configurator	Decoupling component interfaces from their implementations & Reconfiguring components without having to shutdown & restart.
…	…
Broker	Defining the broker's base-line architecture

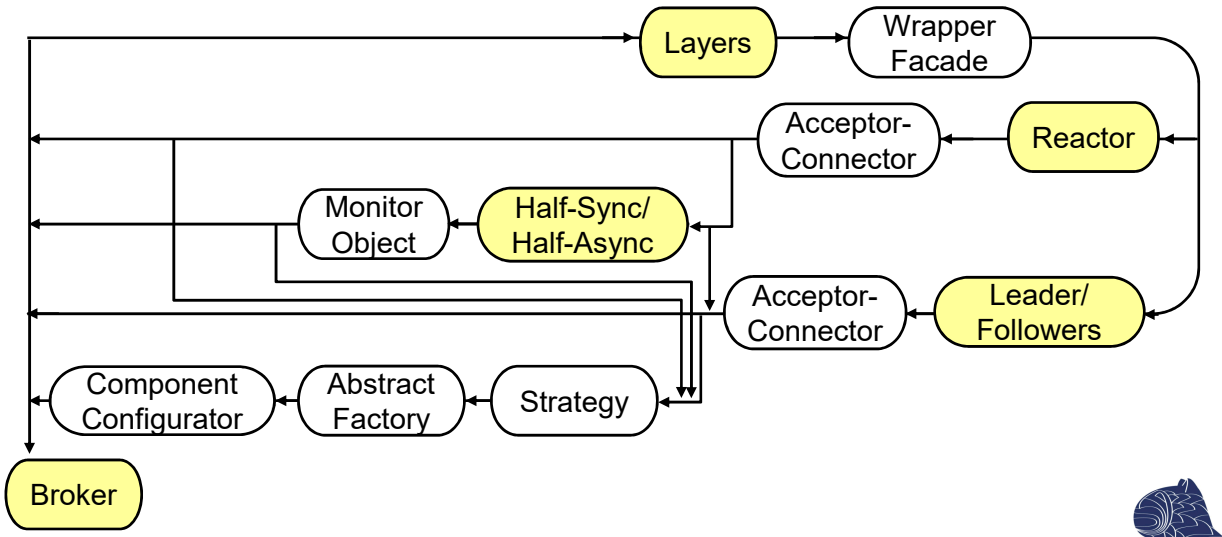
Diseño de un ORB

"Broker pattern language"



Diseño de un ORB

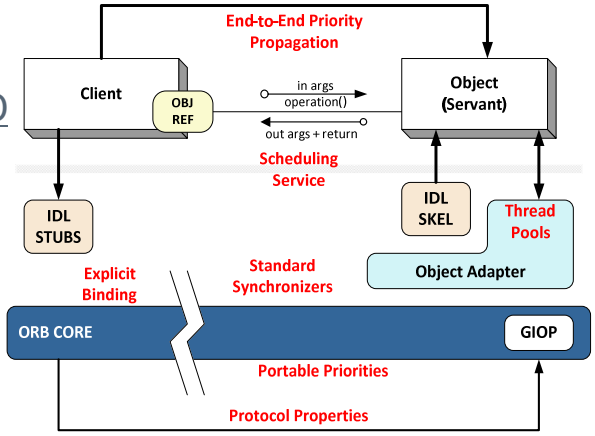
"Broker pattern language"



CORBA

Implementaciones "open source"

- TAO (C++)
  http://www.dre.vanderbit.edu/TAO



- JacORB (Java)
  http://www.jacorb.org

Bibliografía

- Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad & Michael Stal: **Pattern-Oriented Software Architecture. Volume 1: A System of Patterns** Wiley, 1996. ISBN 0471958697