Patrones de Software

Juan Carlos Cerón Barreto
ceron.juan@javeriana.edu.co

Agenda

- Patrones de Comunicación
  - Publisher Subscriber
- Patrones Sistemas Distribuidos
  - Broker

PATRONES DE COMUNICACIÓN

PUBLISHER - SUBSCRIBER


Patrones
Negocio
Arquitectura
Análisis
Diseño
Idiomas
EAI
SOA
Estructurales
Sistemas Distribuidos
…
Estructurales
Control de Acceso
Comunicación
…

Contexto

- Entorno con una infraestructura que conecta aplicaciones (bus,broker,P2P).
- Algunas aplicaciones envían múltiples mensajes.
- Otras aplicaciones están interesadas en diferentes combinaciones de estos mensajes.
- Por ejemplo: manejo de clientes

Problema

- Crear una arquitectura para enviar mensajes a las partes interesadas, sin que haya necesidad de conocer la identidad de los destinatarios.
- Mantener sincronizado el estado de los componentes que se encuentran cooperando en una labor

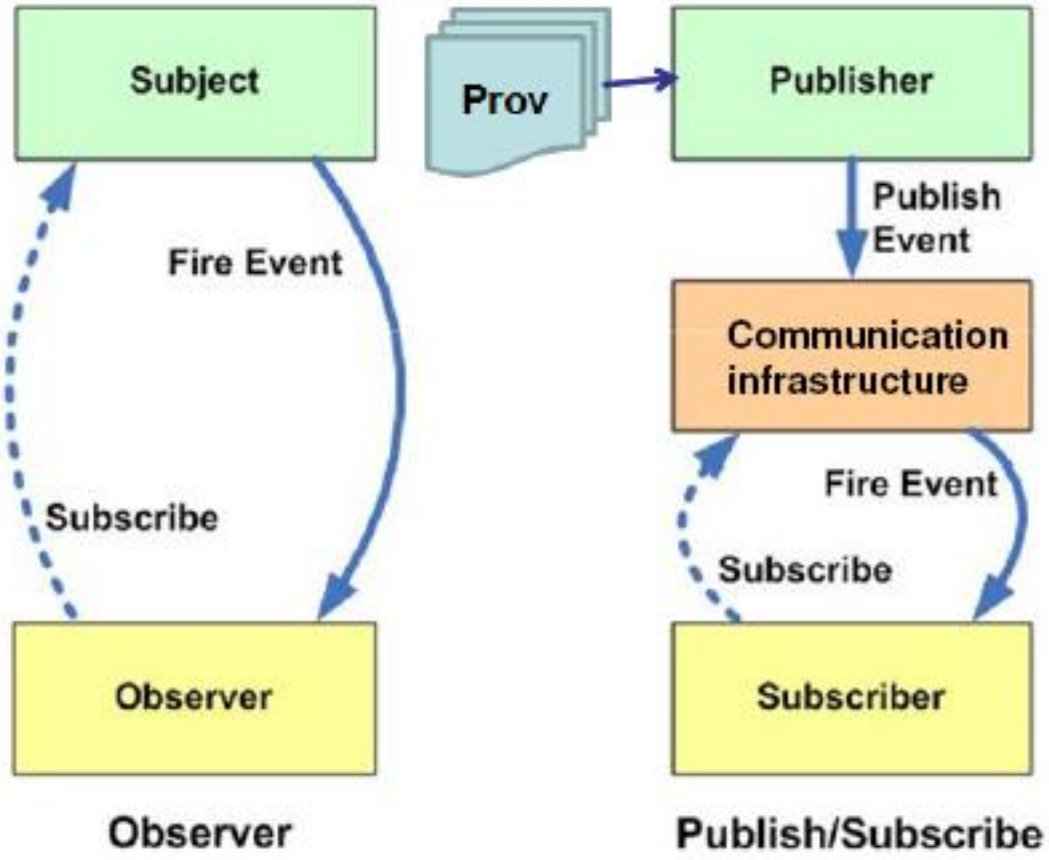
Problema

- **Fuerzas a Balancear**
  - Conocimiento de los destinatarios
  - Tipos de mensajes necesarios por una aplicación
  - Tipos de mensajes que puede enviar una aplicación
  - La medida en que las aplicaciones permiten añadir información a sus mensajes es muy variable
  - Las aplicaciones propietarias pueden generar mensajes poco flexibles o estándares.

Solución

- Especialice a un componente en manejar los cambios de estado.
- Habilite a las aplicaciones a suscribirse y escuchar mensajes en los que están interesadas.
- Establezca un mecanismo de envió de cambios de estado, que sea independiente de las aplicaciones interesadas.
- Establecer una estrategia de suscripción a los cambios de estado

Estructura


Subject
Fire Event
Subscribe
Observer
Observer
Prov
Publisher
Publish Event
Communication infrastructure
Fire Event
Subscribe
Subscriber
Publish/Subscribe

Estructura

- **Generadores de Eventos**
  - Contienen la lógica de negocio que manipula el estado
  - Fuente donde se origina el cambio de estado
  - No dependen de los interesados en los cambios

Estructura

- **Publisher**
  - Expone un API para el registro y desregistro de componentes
  - Mantiene un listado de los componentes interesados en los eventos.
  - Puede decidir cuales cambios se notifican y cuales no
  - Dependiendo del volumen de eventos, puede que necesite encolarlos para posterior procesamiento
  - Envió total o parcial de información del evento (push, pull)

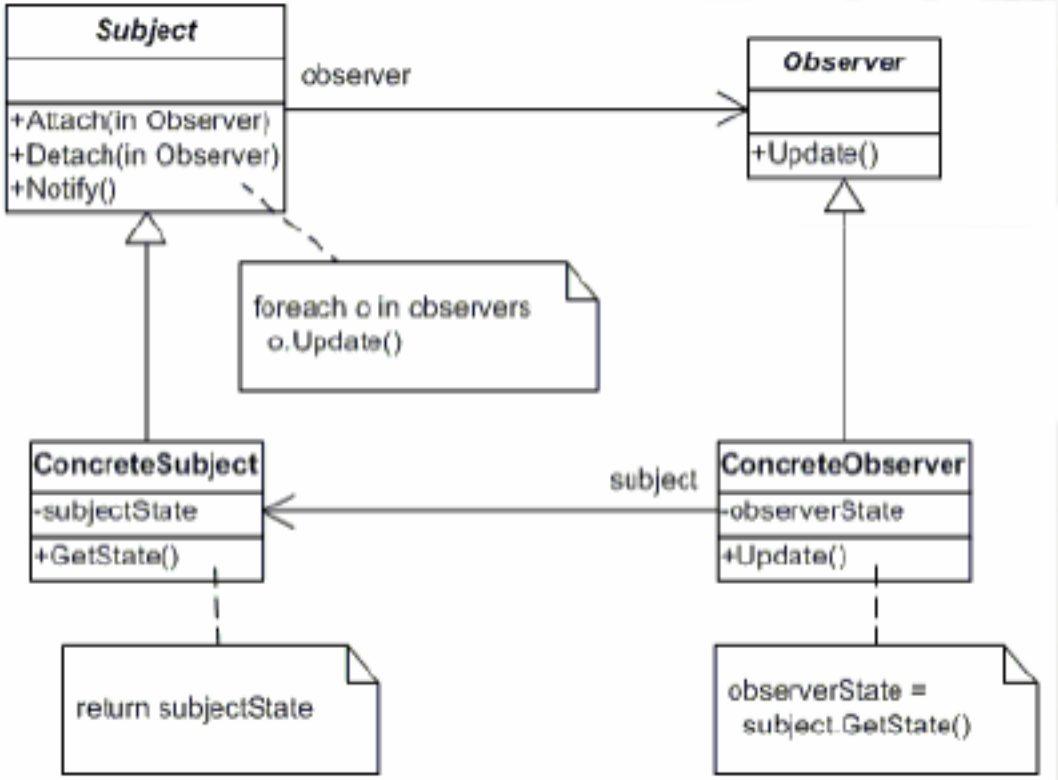
Estructura

- **Infraestructura de Comunicación**
  - Medios de comunicación para informar los cambios.
  - Modelo de dominio de los mensajes a intercambiar.
  - Medios independientes para los proveedores de eventos y para los consumidores

- **Suscriptores**
  - Interesados en los cambios de estados
  - Dependiendo del modelo de notificación pueden tener lógica para la selección de mensajes.

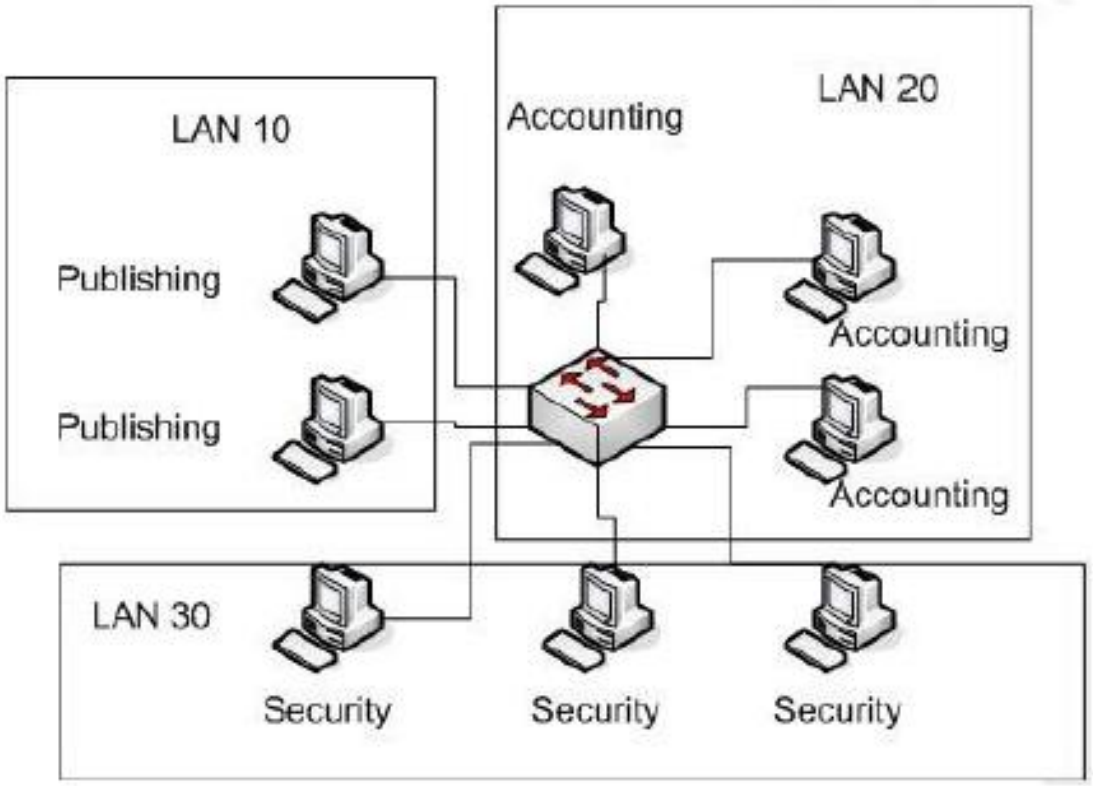
Variantes

Pub/Sub Basado en Listas



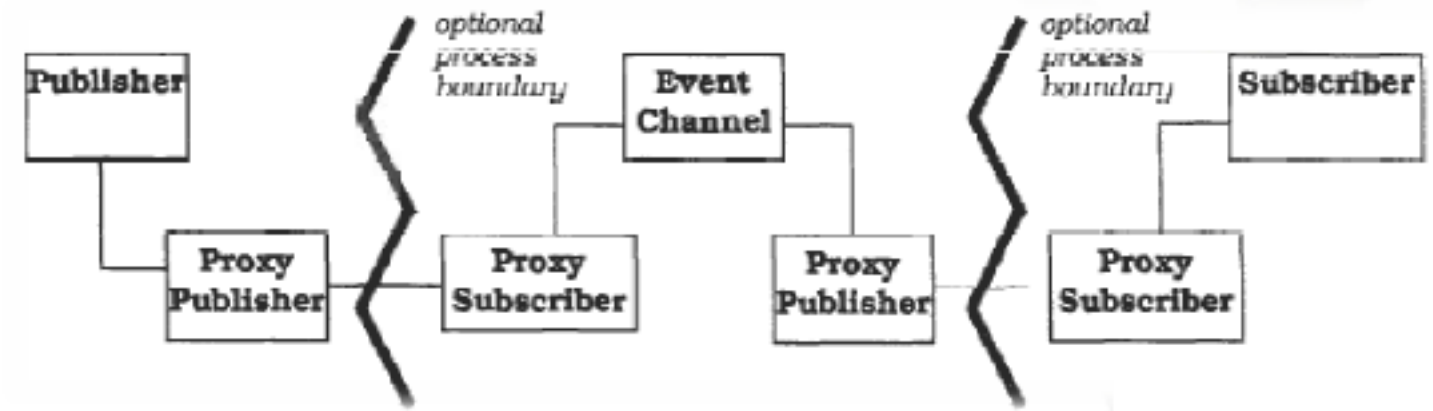
Variantes

Pub/Sub Basado en Broadcast



Variantes

Pub/Sub Basado con canal de eventos

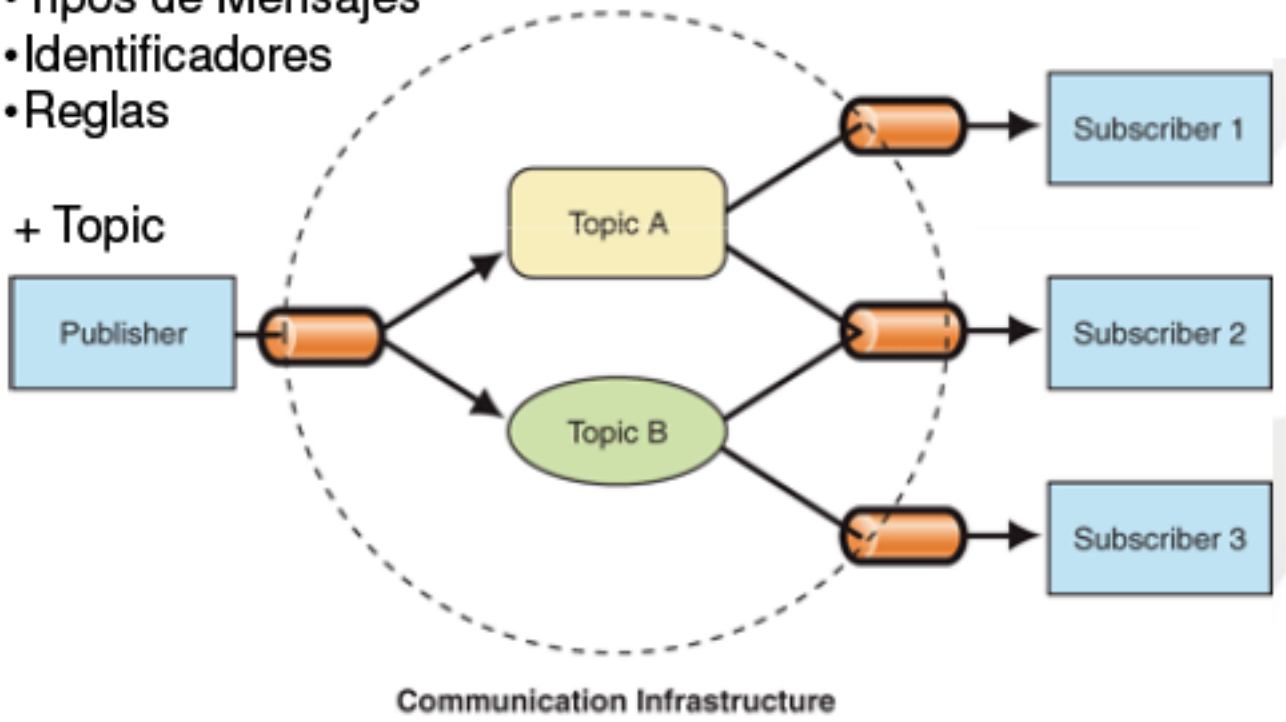


Casos Pub/Sub

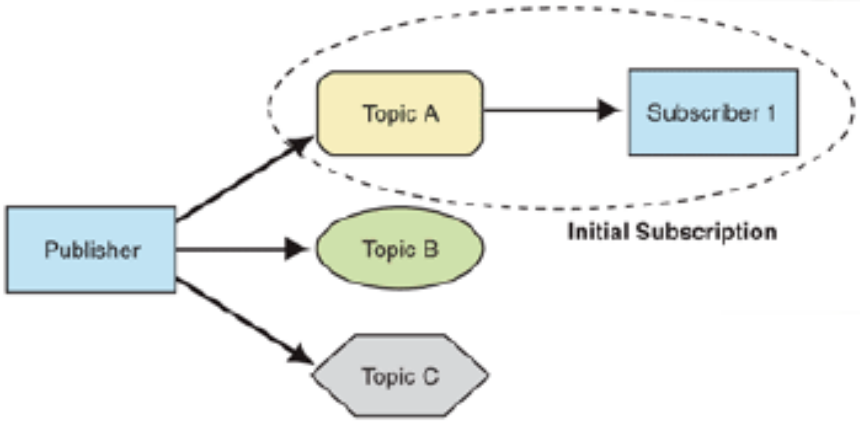
Topic

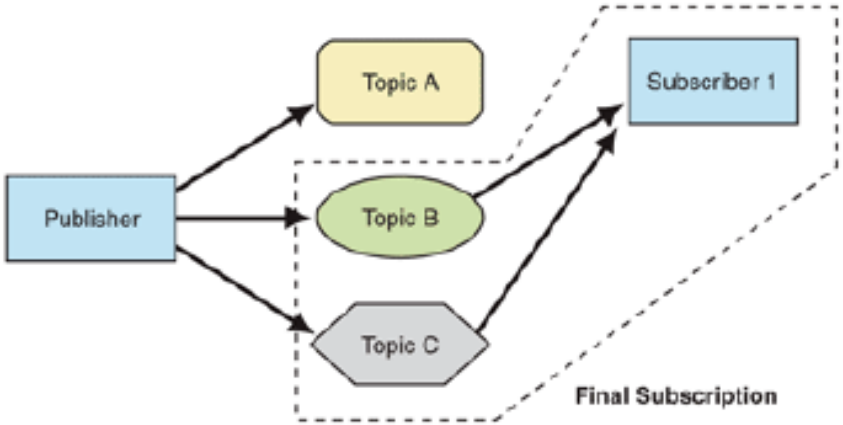
- Tipos de Mensajes
- Identificadores
- Reglas

Msg + Topic



Subscripciones Pub/Sub





Consideraciones Subscripciones

- Suscripción Inicial.
- Suscripciones generales
- Estáticas o dinámicas
- Descubrimiento de tópicos

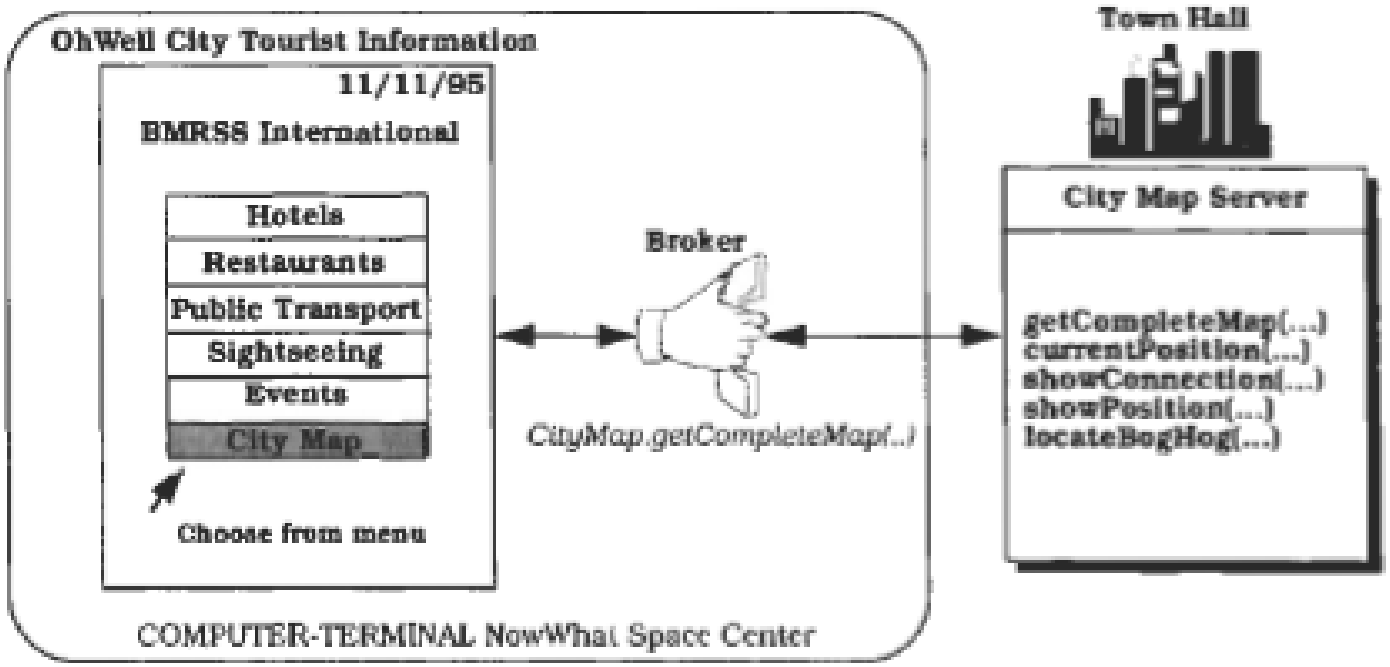
Consecuencias

- Beneficios
  - Bajo acoplamiento.
  - Mejora la seguridad.
  - Mejora la capacidad de pruebas
- Restricciones
  - Aumento complejidad.
  - Aumento en las tareas de mantenibilidad
  - Impacto en el rendimiento

PATRONES
SISTEMAS DISTRIBUIDOS


Patrones
Negocio
Arquitectura
Análisis
Diseño
Idiomas
EAI
SOA
Estructurales
Sistemas Distribuidos
…
Estructurales
Control de Acceso
Comunicación
…

Caso de Estudio



Contexto

Entorno que consiste de múltiples sistemas distribuidos heterogéneos que necesitan interactuar entre ellos de manera sincrónica o asincrónica.

- ¿ Por que tener un entorno distribuido?
  - Capacidad de computo múltiples PCs o Clúster
  - Cierto software sólo está disponible en equipos específicos.
  - Segmentación de red, debido a restricciones de seguridad.
  - Servicios de partners accedidos solo a través de internet

Problema

- Componentes distribuidos necesitan comunicarse (IPC)
- No debe haber diferencia entre modelos centralizados y distribuidos
- Se requiere configuración dinámica de servicios

Problema

- Concurrencia
- Conectividad entre distintas plataformas
- Conexiones de red no confiables.
- Los desarrolladores de aplicaciones no tienen que preocuparse de los detalles de las comunicaciones remotas.

Fuerzas

- Complejidad en la implementación: (Comunicación, codificación y seguridad)
- Modificación de componentes en tiempo de ejecución
- Detalles de la comunicación inter-procesos

Fuerzas

- Flexibilidad en la ubicación de los componentes (“hard-coding”) ..desacoplamiento
- Manejo de recursos
- Transparencia en los tipos de sistemas
- Portabilidad

Solución

Ocultar los detalles de implementación de la invocación de servicios remotos mediante la encapsulación de ellos en una capa que no sea el componente de negocio en sí..” [Buschmann96]

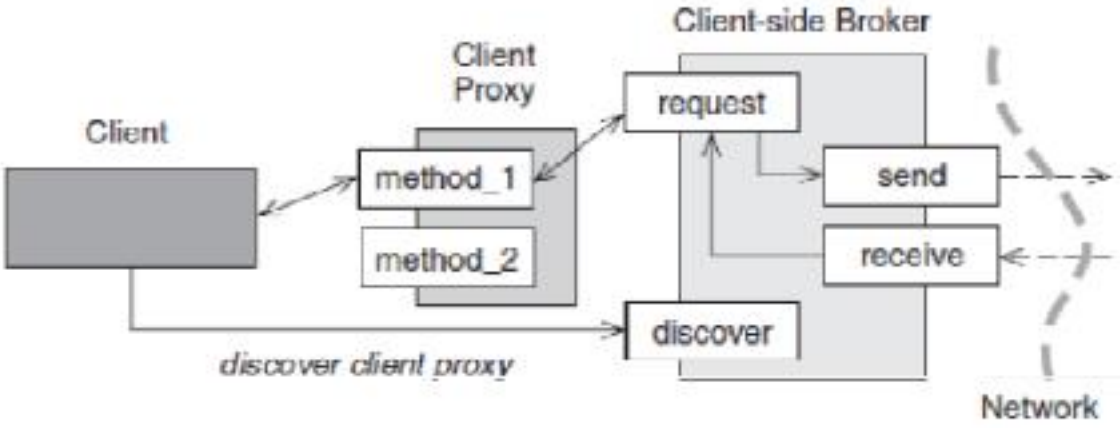
Solución

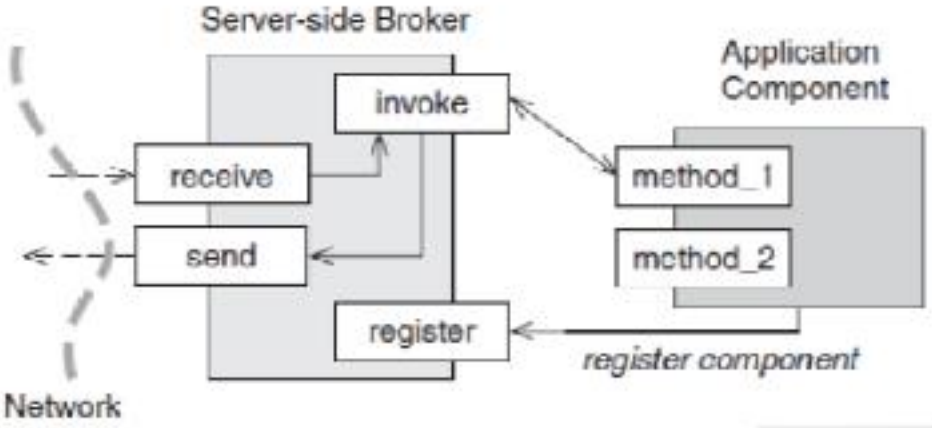
- Interfaz para que el cliente invoque métodos remotos del mismo modo que invoca una interfaz local.
- El objeto del servicio remoto implementa la misma interface usada por el cliente

Solución

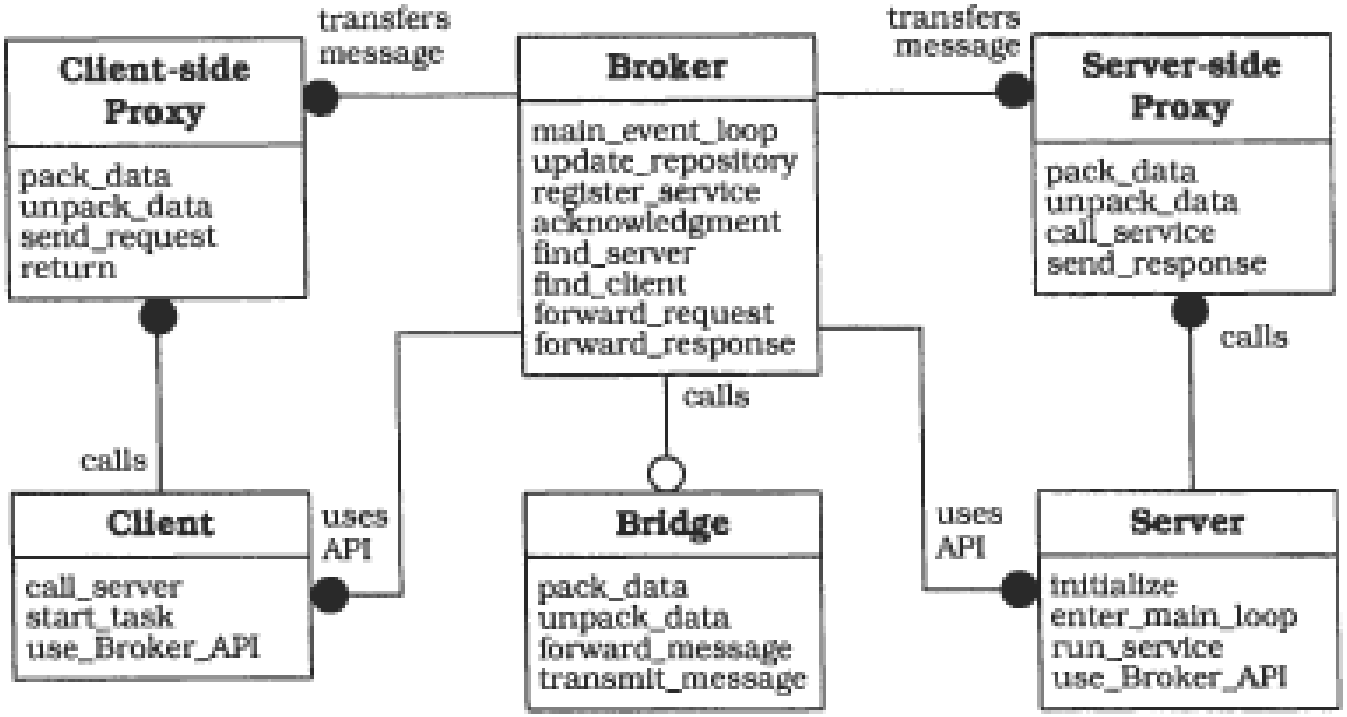
- Incluir un intermediario (Broker)…
  - Desacopla clientes y servidores
  - Encapsula los mecanismos de comunicación
  - Centraliza el acceso a los servicios remotos
  - Une las tecnologías de objetos con las de distribución
  - Permite especializar los equipos de desarrollo

Estructura


Client-side Broker
Client Proxy
Client
request
method_1
send
method_2
receive
discover
discover client proxy
Network


Server-side Broker
Application Component
invoke
receive
method_1
send
method_2
register
register component
Network

Estructura


Client-side Proxy
pack_data
unpack_data
send_request
return
transfers message
Broker
main_event_loop
update_repository
register_service
acknowledgment
find_server
find_client
forward_request
forward_response
transfers message
Server-side Proxy
pack_data
unpack_data
call_service
send_response
calls
calls
calls
Client
call_server
start_task
use_Broker_API
uses API
Bridge
pack_data
unpack_data
forward_message
transmit_message
uses API
Server
initialize
enter_main_loop
run_service
use_Broker_API

Estructura

- **Servidor**
  - Brinda las funcionalidades a los clientes
  - Implementa objetos que expone a través de interfaces
  - Registro con el broker local
  - Envía repuestas y errores a los clientes a través del proxy-server

Estructura

- **Clientes**
  - Aplicaciones que acceden a los servicios de al menos un servidor.
  - Implementa la funcionalidad de usuario
  - Para invocar los servicios, el cliente pasa las solicitudes al broker y recibe la respuesta también del broker.

Estructura

- **Broker**
  - Registrar y desregistrar servidores
  - Transmitir los mensajes
  - Localizar servidores
  - Cuando la solicitud va para un servidor alojado en otro broker, el broker local encuentra una ruta al bróker remoto y le pasa la solicitud.
  - Naming y marshaling se pueden integrar en el broker

Estructura

- **Bridge**
  - Comunicación entre broker
  - Manejo de redes heterogéneas

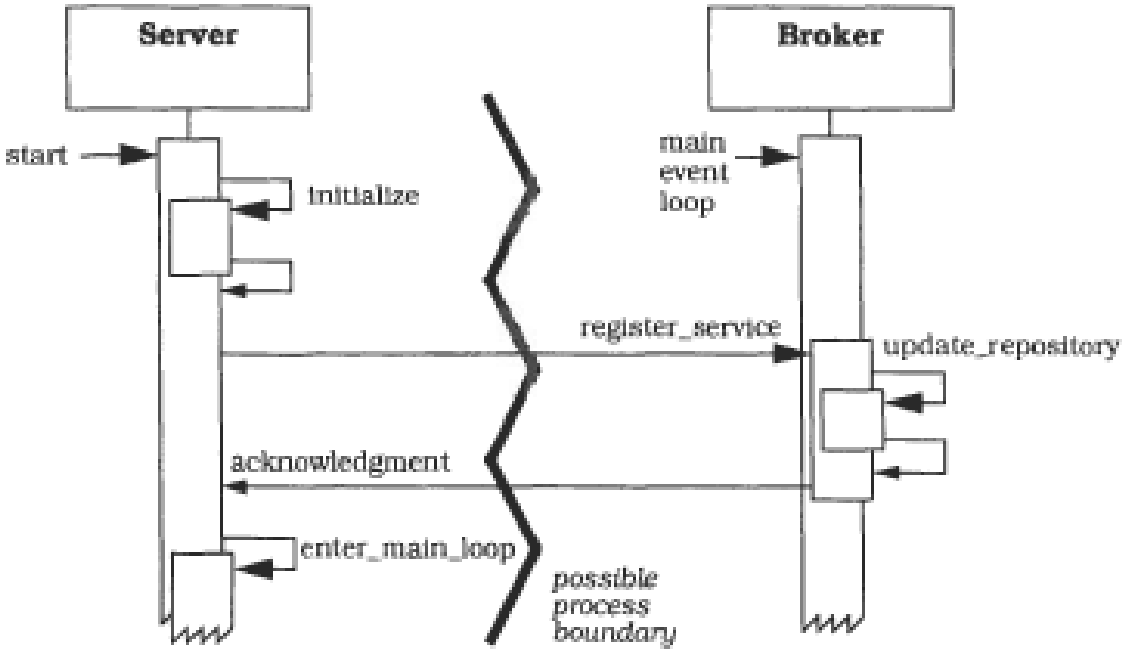
Estructura

- **Proxy Cliente**
  - Proporciona transparencia, hace que el objeto remoto luzca al cliente como un objeto local.
  - Oculta mecanismos de comunicación, manejo de bloques de memoria, mashaling de parámetros y resultados

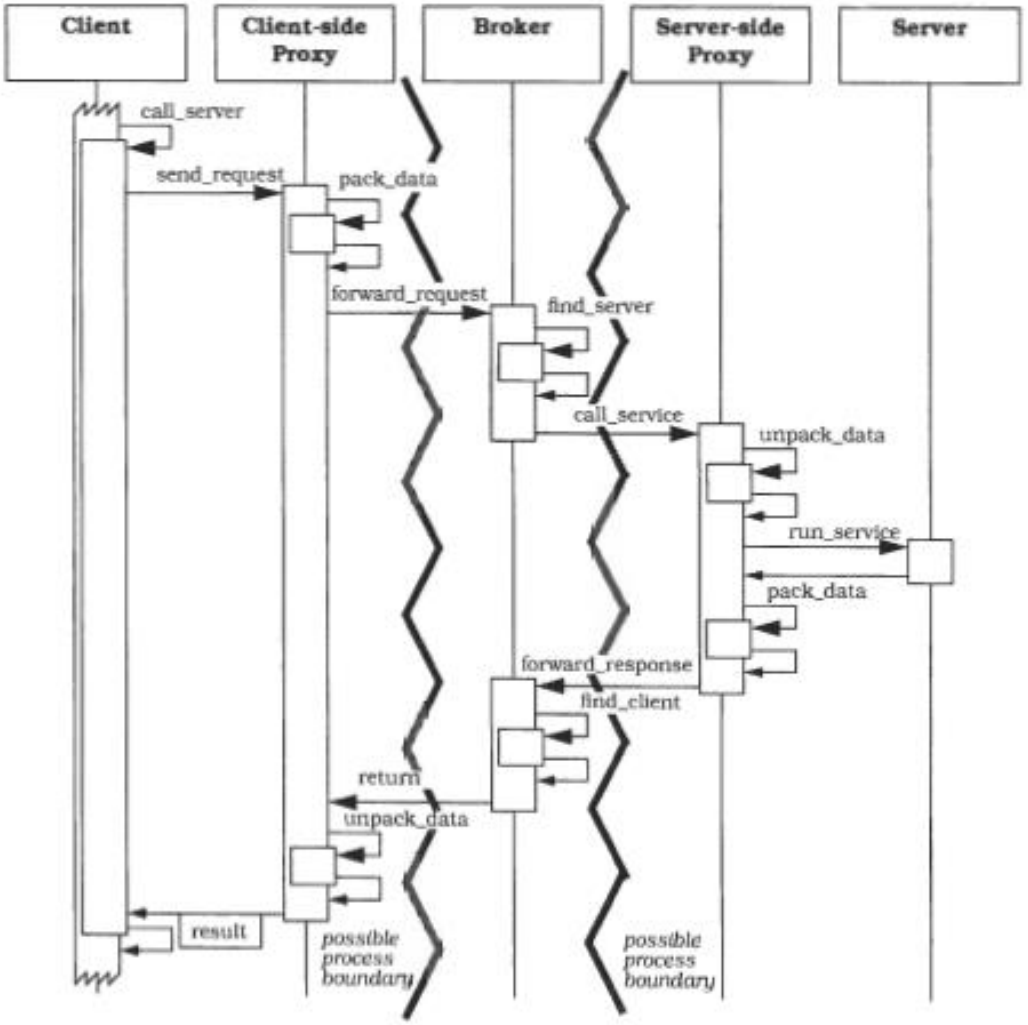
Estructura

- **Proxy Servidor**
  - Análogo al proxy-cliente
  - Además el mashaling de resultados y excepciones

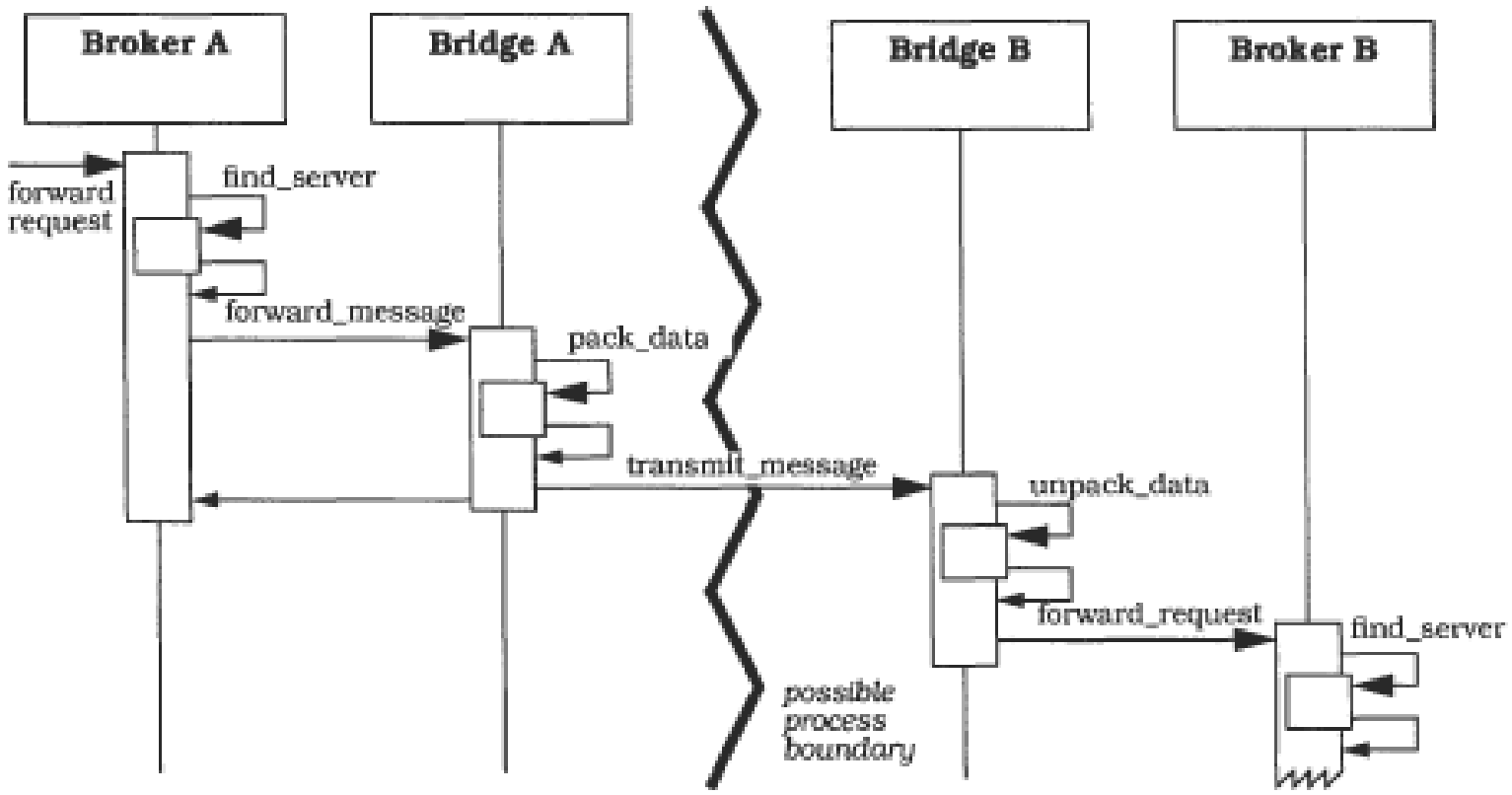
Dinámica


Server
Broker
start
initialize
main
event
loop
register_service
update_repository
acknowledgment
enter_main_loop
possible
process
boundary

Dinámica


Client
Client-side Proxy
Broker
Server-side Proxy
Server
call_server
send_request
pack_data
forward_request
find_server
call_service
unpack_data
run_service
pack_data
forward_response
find_client
return
unpack_data
result
possible process boundary
possible process boundary

Dinámica


Broker A
Bridge A
Bridge B
Broker B
forward request
find_server
forward_message
pack_data
transmit_message
unpack_data
forward_request
find_server
possible process boundary

Implementación

1. Defina un modelo de objetos, o use un modelo existente. ..

- Nombramiento, objetos, solicitudes, valores, excepciones, tipos soportados, interfaces y operaciones.
- Modelo computación de la infraestructura:
  - Estados de objetos de servidor
  - Selección de métodos
  - Ciclo de vida de los objetos.

Implementación

2. Decida el tipo de interoperabilidad entre componentes que el sistema debe ofrecer…
   - Formato binario o lenguaje para definición de interfaces (IDL)
   - Un IDL contiene la descripción textual de las interfaces que un servicio ofrece a sus clientes.
   - El formato binario necesita soporte del lenguaje de programación.

Implementación

3. Especifique las APIs que proporciona el broker…
   - Envió y recepción de solicitudes (sinc o async).
   - Registro de servidores (almacenes de información, generador de identificadores).
   - Mecanismos de IPC eficientes

Implementación

4. Defina los proxies (cliente-servidor)…
   - Especifique el método de generación.
   - Especifique el protocolo de comunicación
   - Defina los mecanismos de marshaling y unmashaling de parámetros.
   - Defina funcionalidades transversales

Implementación

5. Defina el broker
   - Defina los mecanismos de marshaling y unmarshaling de los parámetros si los proxies no lo implementan
   - Si el sistema soporta comunicación sincrónica entre clientes y servidores, proporcione buffers de mensajes.
   - Implemente el servicio de directorios
   - Implemente el servicio de nombres cuando lo requiera
   - En el caso de invocación dinámica de métodos cree un medio para mantener la información de tipos
   - Considere escenarios de error (un componente falla, la comunicación falla)

Variantes

1. Indirect vs. direct communication
2. Message Passing Broker
3. Trader System
4. Adapter Broker System
5. Callback Broker

Consecuencias

- Beneficios
  - Aislamiento.
  - Simplicidad.
  - Flexibilidad
- Restricciones
  - Optimizaciones limitadas.
  - Controles de acceso
  - Manejo de errores a bajo nivel
  - Punto de fallo critico
  - Pruebas y depuración

Bibliografía

- Pattern-Oriented Software Architecture: A System of Patterns, F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal. John Wiley & Sons, 1996
- Patterns of Enterprise Application Architecture: Martin Fowler, Addison-Wesley Professional, 1 edition ,November 15, 2002.

Preguntas