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

Middleware – Publish/Subscribe

© Fernando Berzal, berzal@acm.org

Middleware – Publish/Subscribe

- Motivación:
  Problemas de interés práctico en sistemas distribuidos.
- Forwarder-Receiver
- Dispatcher
- Publisher-Subscriber
- DDS [Data Distribution Service]

Sistemas distribuidos

Motivación

Problemas de interés práctico

- Portabilidad
- Modularidad
- Transparencia (localización)
- Consistencia (comunicación entre componentes).

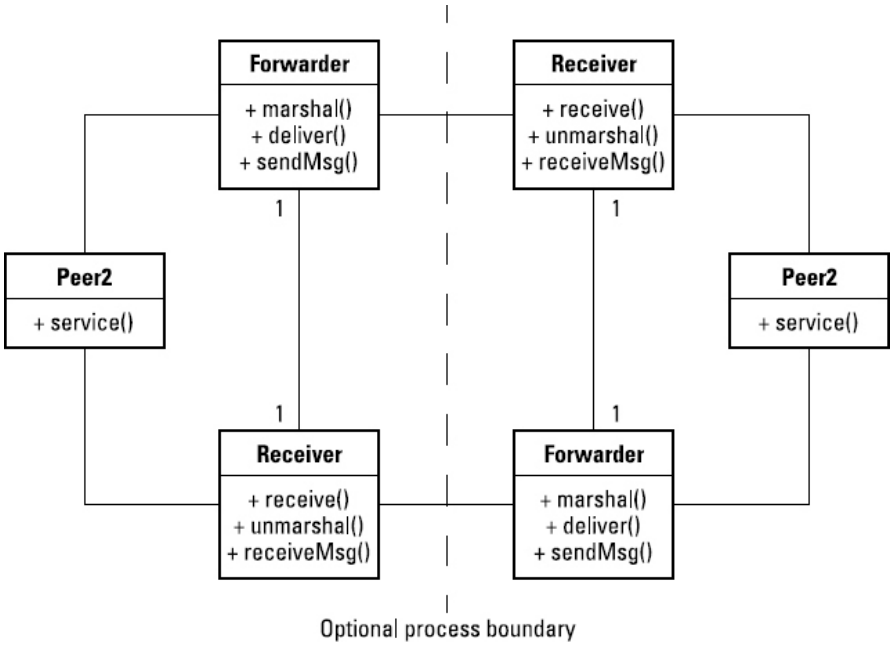
Sistemas distribuidos

¿Cómo resolver esos problemas prácticos?

Más patrones de diseño…

- **Forwarder-Receiver** (transparencia y portabilidad encapsulando los detalles de la comunicación).
- **Client-Dispatcher-Server** (capa intermedia para conectar cliente y servidor de forma transparente).
- **Publisher-Subscriber** (consistencia en la comunicación entre componetnes).

Forwarder-Receiver



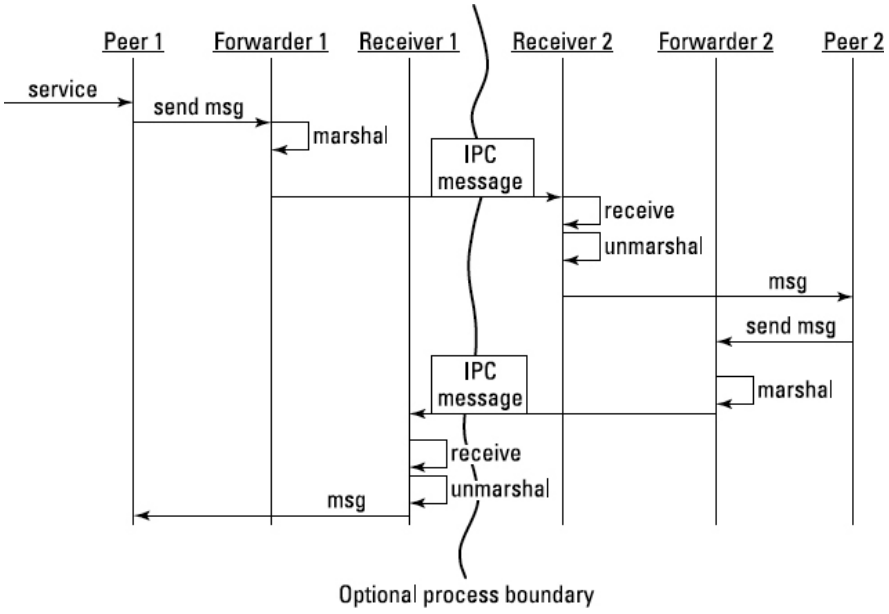
Forwarder-Receiver

Class Forwarder	***Collaborators***
Responsibility • Provides a general interface for sending messages. • Marshals and delivers messages to remote receivers. • Maps names to physical addresses.	• Receiver

Class Receiver	***Collaborators***
Responsibility • Provides a general interface for receiving messages. • Receives and unmarshals messages from remote forwarders.	• Forwarder

Class Peer	***Collaborators***
Responsibility • Provides application services. • Communicates with other peers.	• Forwarder • Receiver

Forwarder-Receiver



Forwarder-Receiver

Implementación

1. Definir la correspondencia entre nombres y direcciones.
2. Definir el protocolo de mensajes (p.ej. time outs).
3. Elegir un mecanismo de comunicación (IPC).
4. Construir el emisor [forwarder].
5. Construir el receptor [receiver]: blocking vs. nonblocking.
6. Implementar los pares en la aplicación.

Forwarder-Receiver

Beneficios

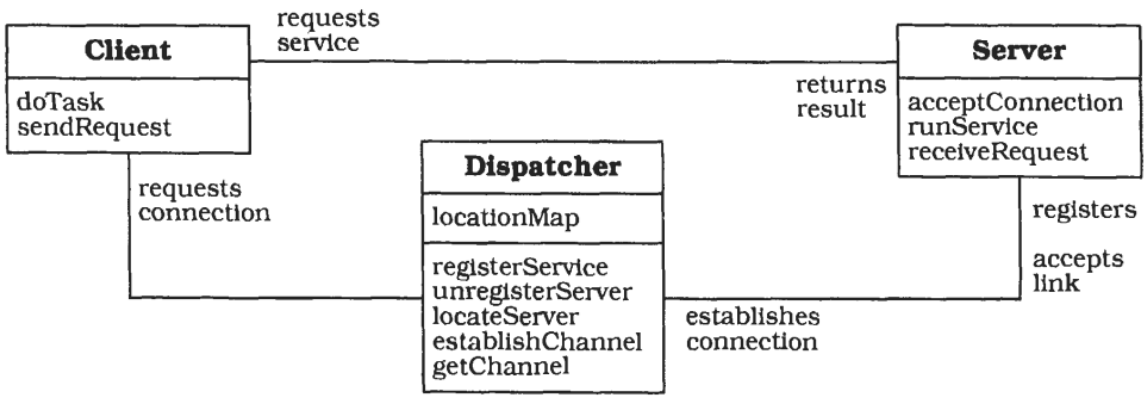
- Comunicación eficiente entre procesos.
- Encapsulación del código asociado a IPC.

Limitación

- No soporta la reconfiguración flexible de componentes.

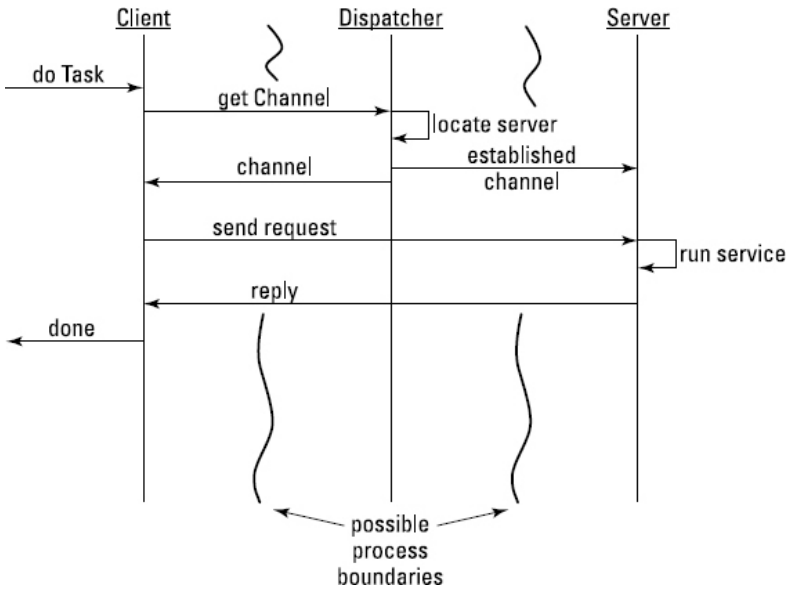
Dispatcher

≈ Broker con comunicación directa



Dispatcher

≈ Broker con comunicación directa



Dispatcher

Implementación

1. Descomposición entre clientes y servidores.
2. Mecanismo de comunicación (IPC vs. shared memory).
3. Protocolo de interacción entre componentes.
4. Servicio de nombres @ Dispatcher.
5. Construir el dispatcher.
6. Implementar clientes y servidores.

Dispatcher

Beneficios

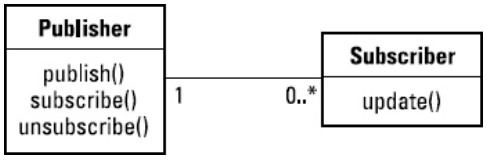
- Posibilidad de intercambiar/añadir servidores.
- Transparencia (localización y migración).
- Reconfiguración.
- Tolerancia a fallos.

Limitaciones

- Menor eficiencia (nivel de indirección).
- Sensibilidad a cambios en la interfaz del dispatcher.

Publisher-Subscriber

Variante del patrón de diseño OBSERVER

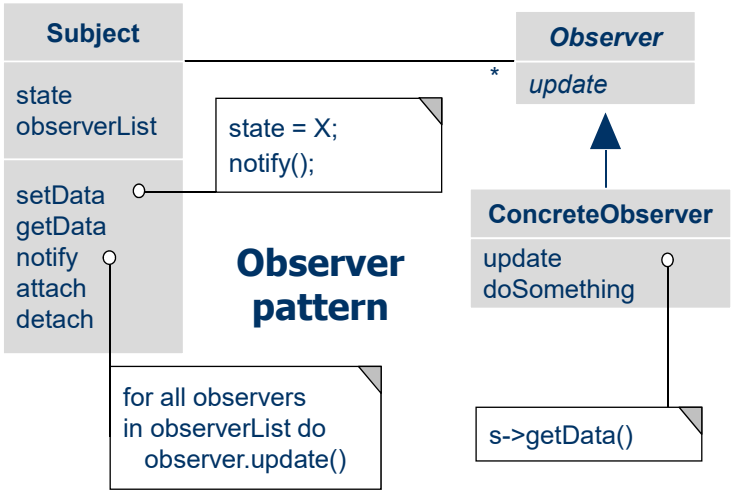


Mantiene sincronizado el estado de los componentes:

El editor [publisher] notifica a sus suscriptores [subscribers] de los cambios que se producen.

Publisher-Subscriber

RECORDATORIO: El patrón de diseño OBSERVER



Publisher-Subscriber

Component	Responsibilities	Collaborations
Communication infrastructure	Maintains the subscribers' subscriptions. Inspects the topic-related information or the content information that is included in each published message. Transports the message to the subscribed applications.	The publisher publishes messages. The subscriber subscribes to topics and receives messages.
Publisher	Inserts topic-related information or content information in each message. Publishes the message to the communication infrastructure.	The communication infrastructure transports messages to subscribers.
Subscriber	Subscribes to one or more topics or message content types. Consumes messages published to the subscribed topics.	The communication infrastructure transports published messages from the publisher.

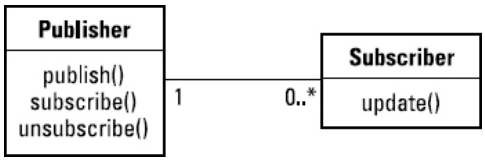
Publisher-Subscriber

Implementación

1. Definir las políticas de publicación:
   - Todas las notificaciones a todos los subscriptores vs. algunas notificaciones a algunos subscriptores (notificaciones selectivas).
   - Actualizaciones completas vs. Avisos de cambios.
   - Actualizaciones inmediatas vs. Lotes.

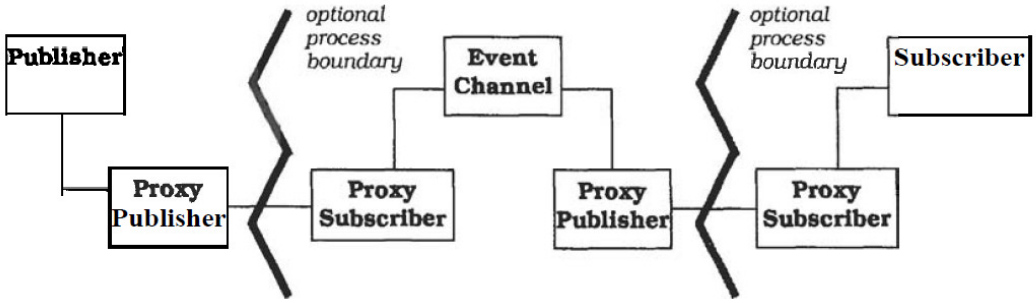
Publisher-Subscriber

Implementación



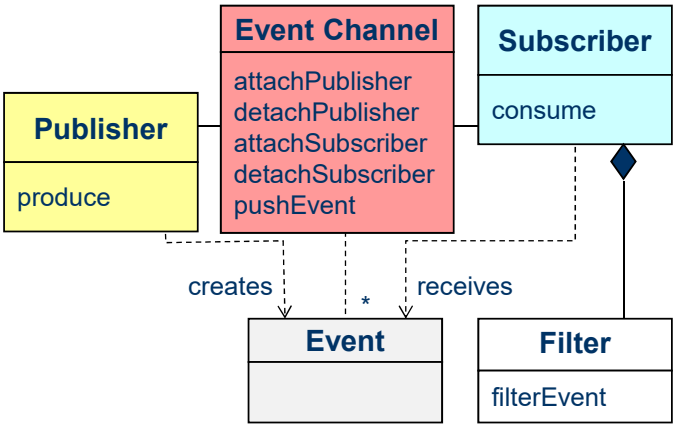
2. Interfaz del editor [publisher]:
   - **subscribe():** Realización de suscripciones.
   - **unsubscribe():** Anulación de suscripciones.
3. Interfaz del suscriptor [subscriber]:
   - **update():** Recepción de actualizaciones.

Publisher-Subscriber



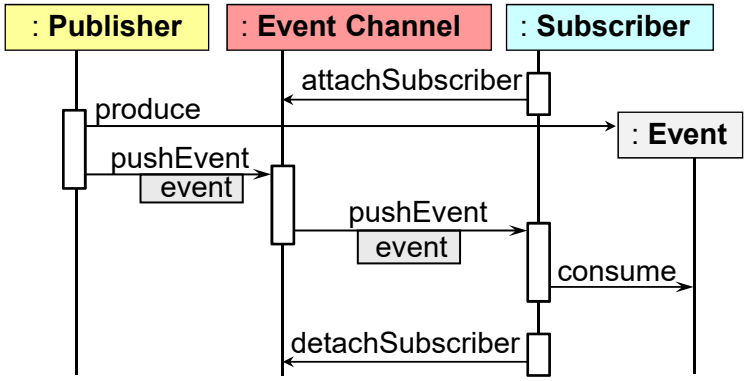
Publisher-Subscriber

Estructura



Publisher-Subscriber

Dinámica



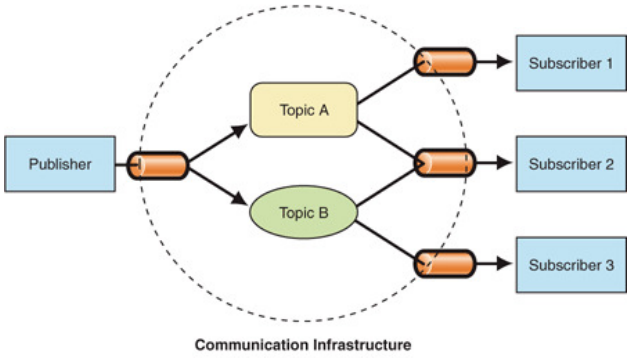
Publisher-Subscriber

Decisiones de diseño

- Modelo de notificación de eventos (push/pull)
- Estrategias de entrega de mensajes
  (basadas en prioridades vs. FIFO [first-in/first-out])
- Filtrado de mensajes

Publisher-Subscriber

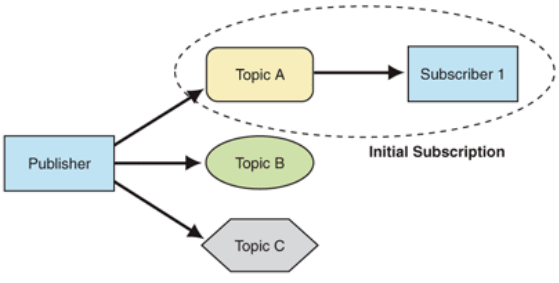
Subscripción selectiva
por temas [topics]

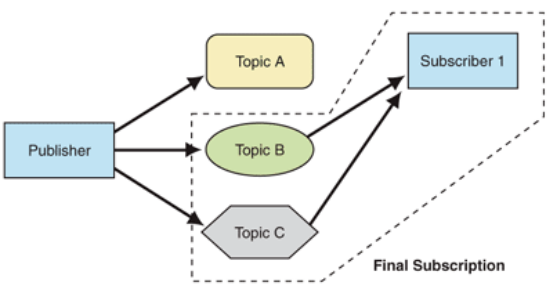


Microsoft®
patterns & practices
proven practices for predictable results

Publisher-Subscriber

Subscripción dinámica





Microsoft®
patterns & practices
proven practices for predictable results

Publisher-Subscriber

Beneficios

- Acoplamiento débil.
- Escalabilidad

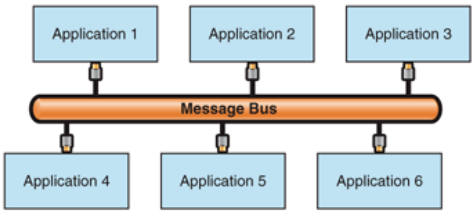
Limitaciones

- Inflexibilidad a la hora de introducir cambios en la estructura de los datos publicados (hay que cambiar todos los posibles suscriptores o mantener la compatibilidad hacia atrás).
- Problemas en la entrega de mensajes (p.ej. QoS).

Publisher-Subscriber

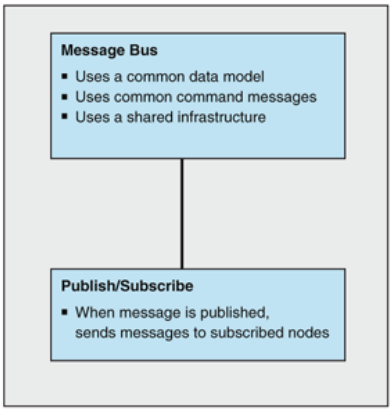
Topología

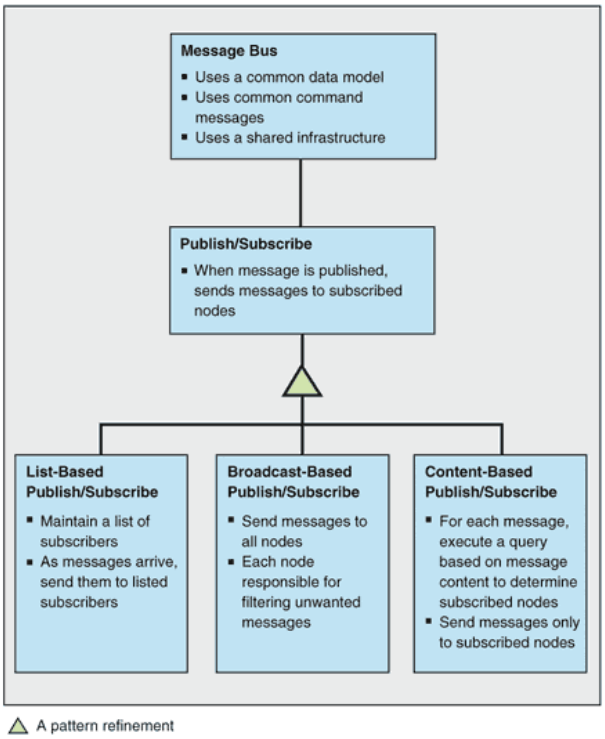
- Message broker (store&forward, prioridades)
- Message/event bus



EJEMPLO:
DDS [Data Distribution Service] utiliza IP multicasting

Publisher-Subscriber

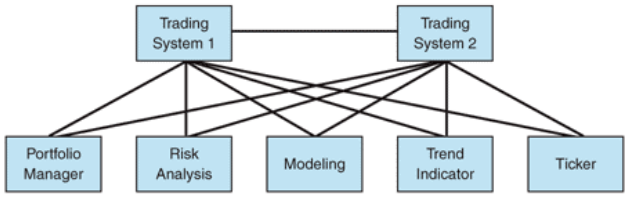


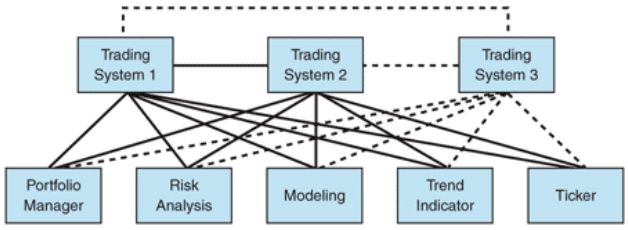


Publisher-Subscriber

Ejemplo

Conexiones punto a punto…

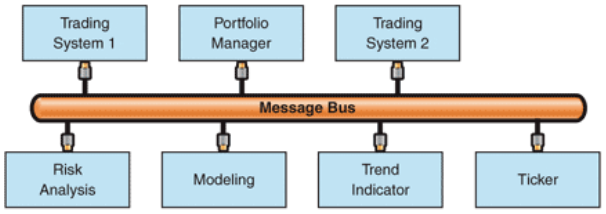


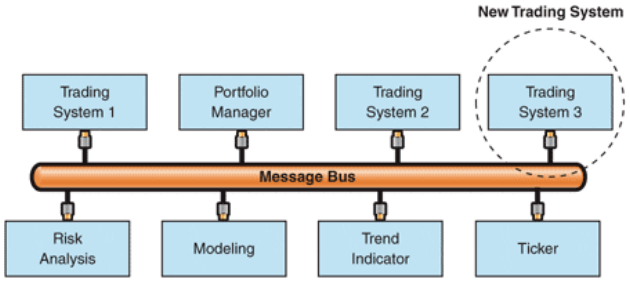


Publisher-Subscriber

Ejemplo

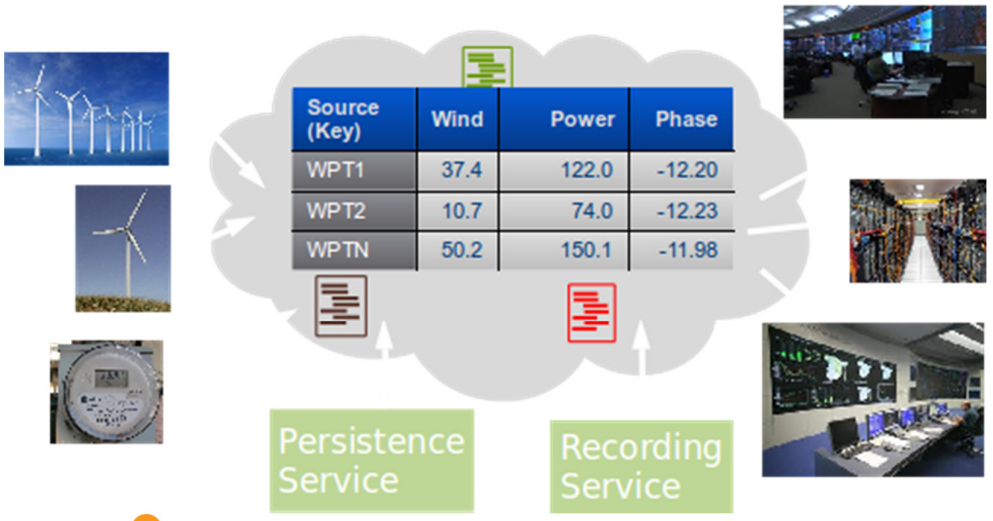
Mejor con un bus de mensajes





DDS [Data Distribution Service]

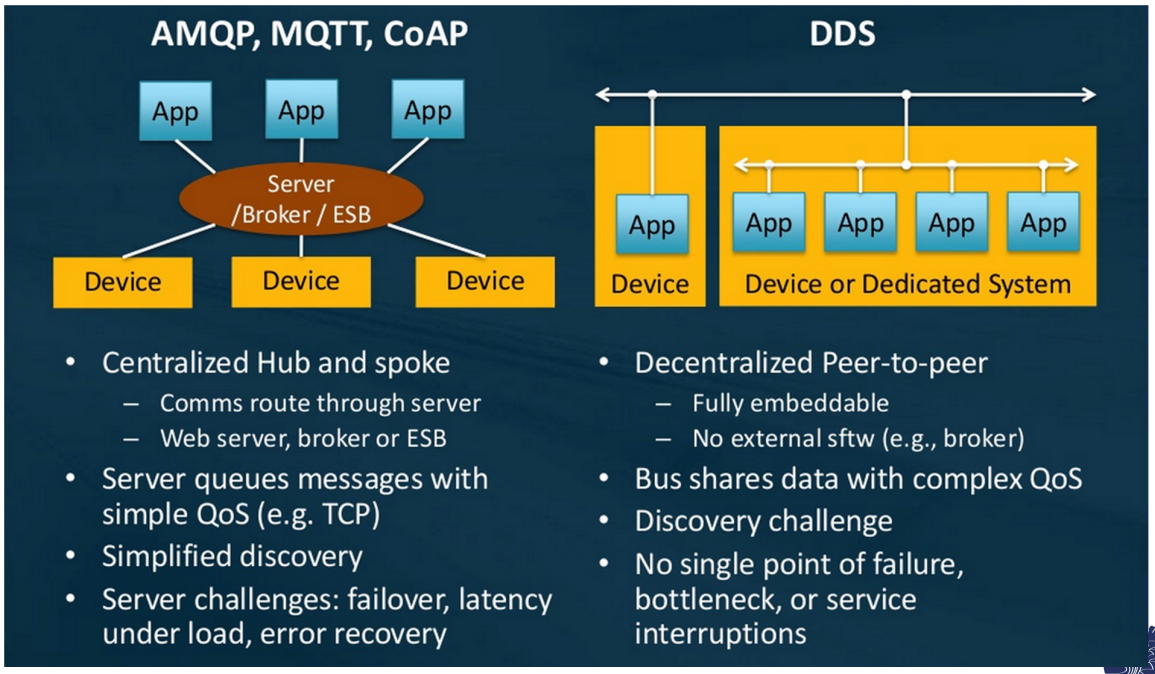
DATA-CENTRIC MIDDLEWARE



rti

OMG
OBJECT MANAGEMENT GROUP

DDS [Data Distribution Service]

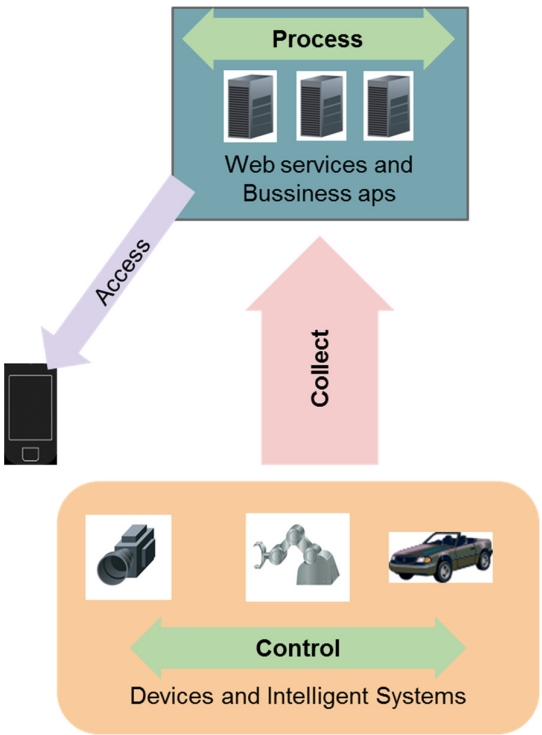


- Centralized Hub and spoke
  - Comms route through server
  - Web server, broker or ESB
- Server queues messages with simple QoS (e.g. TCP)
- Simplified discovery
- Server challenges: failover, latency under load, error recovery

- Decentralized Peer-to-peer
  - Fully embeddable
  - No external sftw (e.g., broker)
- Bus shares data with complex QoS
- Discovery challenge
- No single point of failure, bottleneck, or service interruptions

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

DDS [Data Distribution Service]



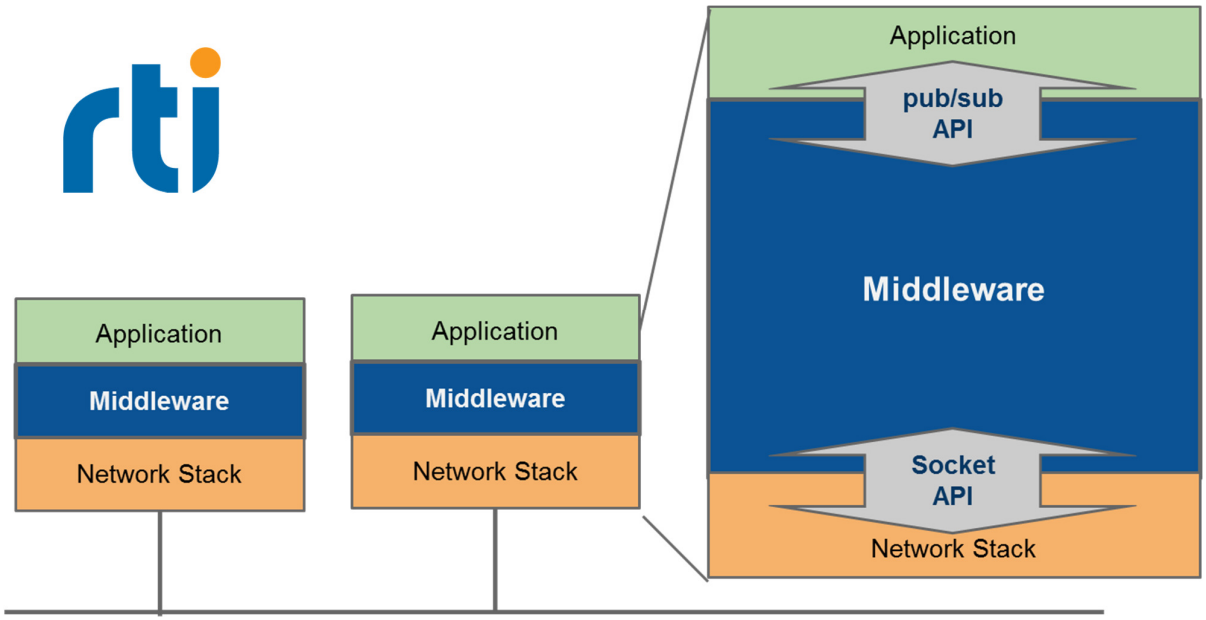
Access
- Link sparse endpoints
- **XMPP**

Process
- Biz intelligence
- Centralized/ESB
- ~100ms
- **MQ/AMQP**

Collect
- Collect data
- Hub & spoke
- ~10ms
- **MQTT/CoAP**

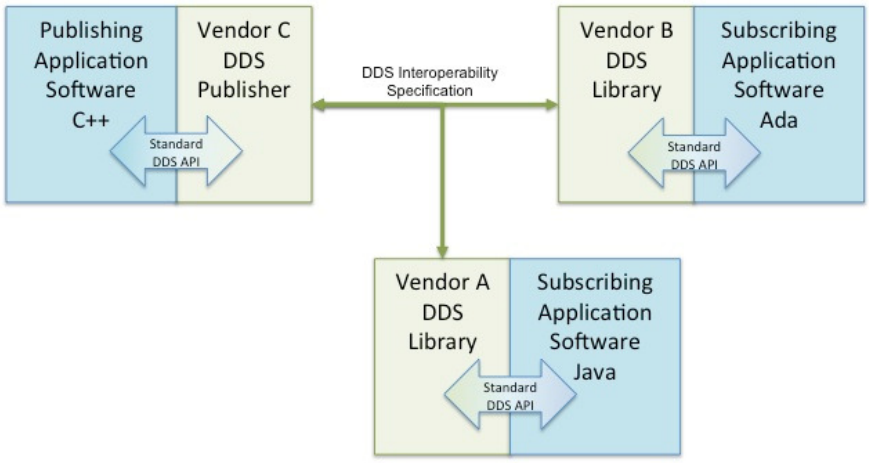
Control, distribute
- DataBus
- ~.01ms
- **DDS**

DDS [Data Distribution Service]



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

DDS [Data Distribution Service]

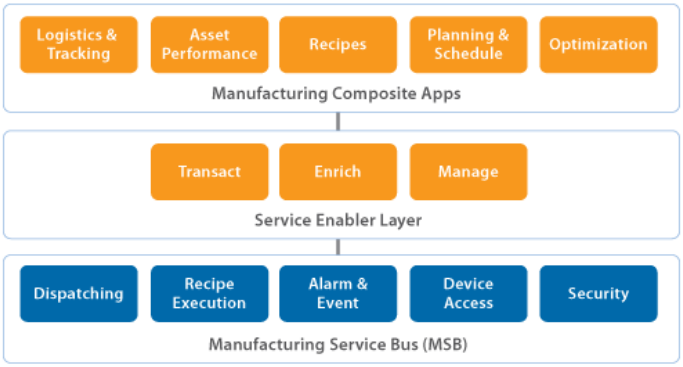


https://en.wikipedia.org/wiki/Data_Distribution_Service

DDS [Data Distribution Service]

Caso práctico

Manufacturing Service Bus

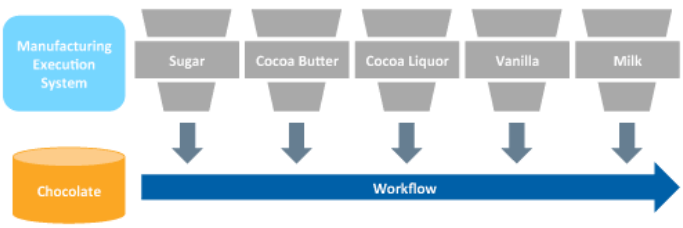


http://www.rti.com/resources/usecases/chocolate-factory.html

DDS [Data Distribution Service]

Caso práctico

Manufacturing Service Bus: Chocolate Factory

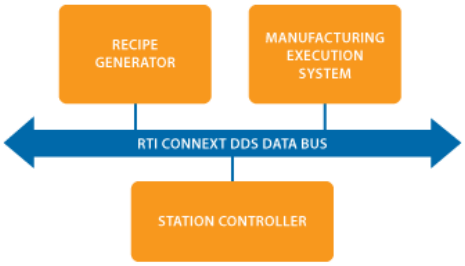


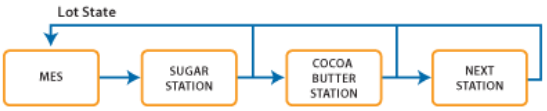
http://www.rti.com/resources/usecases/chocolate-factory.html

DDS [Data Distribution Service]

Caso práctico

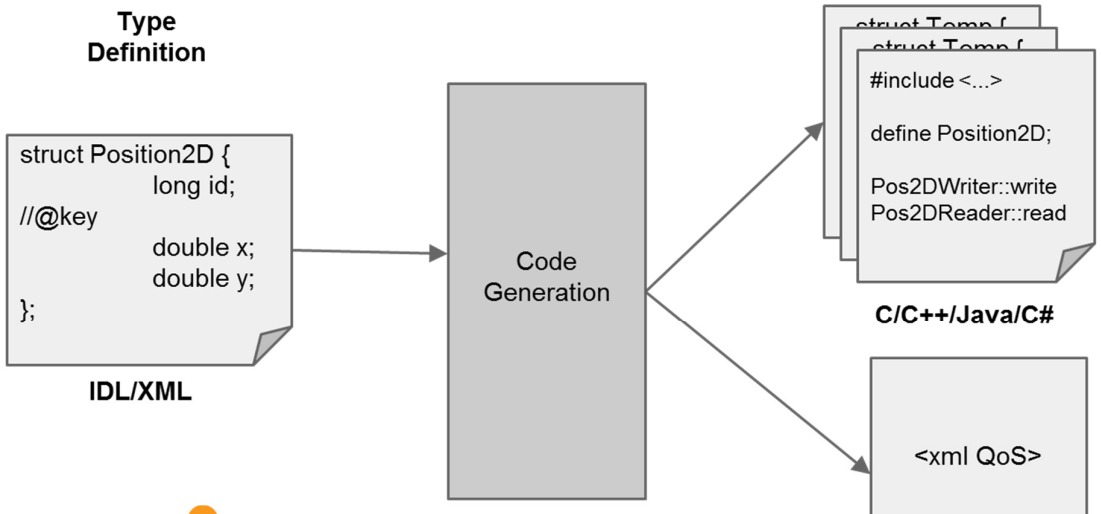
Manufacturing Service Bus: Chocolate Factory



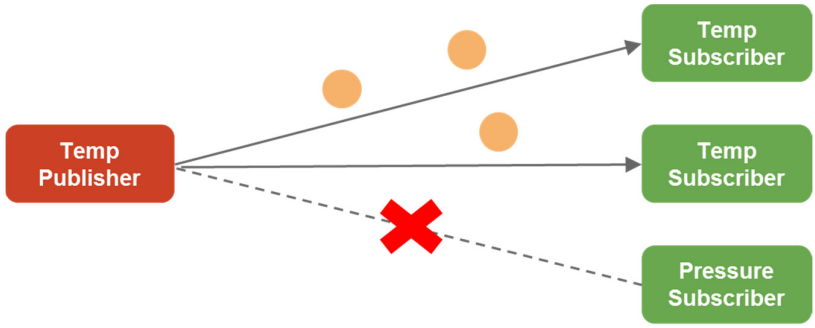


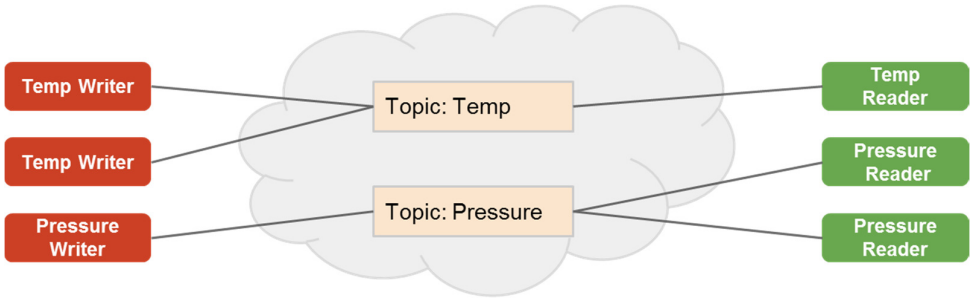
http://www.rti.com/resources/usecases/chocolate-factory.html

DDS [Data Distribution Service]



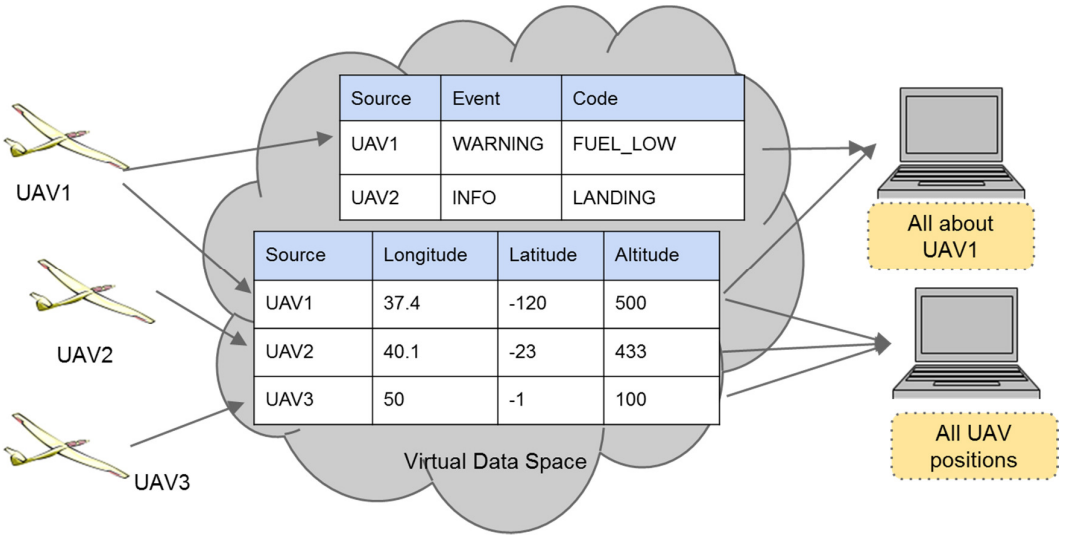
DDS [Data Distribution Service]





rti Your systems. Working as one.

DDS [Data Distribution Service]



rti Your systems. Working as one.

DDS [Data Distribution Service]

Publisher

```
connector = rti.Connector("MyParticipantLibrary::Sensor", 'Tutorial.xml')
writer = connector.getOutput("TempPublisher::TempWriter")
writer.instance.setString('id', sensor.id)
writer.write()
```

Subscriber

```
connector = rti.Connector("MyParticipantLibrary::Sensor", 'Tutorial.xml')
reader = connector.getInput("TempPublisher::TempWriter")
reader.read()
for i in nsamples
  if reader.infos.isvalid(i)
    ample = reader.samples.getDictionary(i)
```

rti Your systems. Working as one.

DDS [Data Distribution Service]

Configuración XML

1. QoS [Quality of Service]

rti Your systems. Working as one.

```
<qos_library name="QosLibrary">
    <qos_profile name="DefaultProfile“ is_default_qos="true">
      <participant_qos>
        <transport_builtin>
          <mask>SHMEM</mask>  <!-- <mask>UDPV4</mask>-->
        </transport_builtin>
      </participant_qos>
      <datareader_qos>
        <!-- Modify reader values here -->
      </datareader_qos>
    </qos_profile>
  </qos_library>
```

DDS [Data Distribution Service]

Quality of Service	Quality of service
DURABILITY	USER_DATA
HISTORY	TOPIC_DATA
READER DATA LIFECYCLE	GROUP_DATA
WRITER DATA LIFECYCLE	PARTITION
LIFESPAN	PRESENTATION
ENTITY FACTORY	DESTINATION ORDER
RESOURCE LIMITS	OWNERSHIP
RELIABILITY	OWNERSHIP STRENGTH
TIME BASED FILTER	LIVELINESS
DEADLINE	LATENCY BUDGET
CONTENT FILTERS	TRANSPORT PRIORITY

Volatility
Infrastructure
Delivery
User
Presentation
Redundancy Transport

DDS [Data Distribution Service]

Configuración XML

2. Definición de tipos

rti Your systems. Working as one.

```
<types>
  <struct name="Sensor" extensibility="extensible">
   <member name="id" stringMaxLength="128" id="0" type="string“
        key="true"/>
   <member name="value" id="1" type="long"/>
   <member name="timestamp" id="2" type="long"/>
 </struct>
</types>
```

```
struct Sensor {
        string id; //@key
        long value;
        long timestamp;
};
```

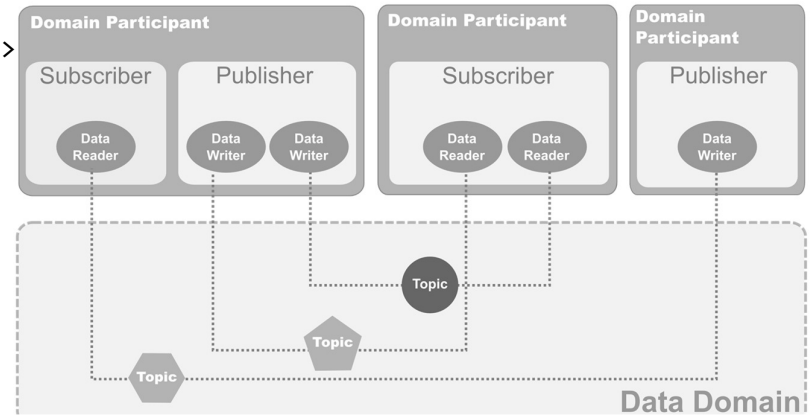
DDS [Data Distribution Service]

Configuración XML

3. Domains & Topics

Your systems. Working as one.

```
<domain_library name="MyDomainLibrary">
  <domain name="MyDomain" domain_id="0">
    <register_type name="Sensor" kind="dynamicData" type_ref="Sensor"/>
    <topic name="Temperature" register_type_ref="Sensor"/>
  </domain>
</domain_library>
```



DDS [Data Distribution Service]

Configuración XML

4. Entidades participantes

Your systems. Working as one.

```
<participant_library name="MyParticipantLibrary">
 <domain_participant name="Console"
        domain_ref="MyDomainLibrary::MyDomain">
   <subscriber name="TempSubscriber">
     <data_reader name="TempReader" topic_ref="Temperature"/>
   </subscriber>
  </domain_participant>
  <domain_participant name="Sensor"
        domain_ref="MyDomainLibrary::MyDomain">
  <publisher name="TempPublisher">
    <data_writer name="TempWriter" topic_ref="Temperature"/>
    </publisher>
  </domain_participant>
</participant_library
```

DDS [Data Distribution Service]

Uso

Your systems. Working as one.

- Filtrado (por contenido o tiempo).
- Configuración QoS:
  - Disponibilidad [availability]: liveliness/ownership.
    p.ej. sensor de respaldo (si falla el primario).
  - Durabilidad [durability]
    p.ej. historia reciente de los datos del sensor.
  - Particionamiento [data isolation/partition]
    p.ej. actualizaciones sólo de determinadas zonas.

DDS [Data Distribution Service]

Uso: Problemas en la entrega de mensajes
https://blogs.rti.com/2016/03/02/where-is-my-data/

Your systems. Working as one.

- Proceso de descubrimiento
  Un DataReader puede perder las primeras muestras de un DataWriter si éste no ha descubierto al DataReader en el momento de publicar losprimeros datos (posible solución: mantener las muestras usando DurabilityQosPolicy=TRANSIENT_LOCAL).
- Comunicación fiable
  Aunque la comunicación sea fiable [ReliabilityQosPolicy=RELIABLE], en una red congestionada se pueden perder muestras si no las mantenemos en caché [HistoryQosPolicy=KEEP_ALL].
- DataReader marcado como inactivo
  Un DataReader se considera inactivo si no envía mensajes ACK/NACK en respuesta a n mensajes periódicos [heartbeat] del DaraWriter (parámetros max_heartbeat_retries & HB period).

DDS [Data Distribution Service]

Agradecimientos

Aída Jiménez Moscoso del Prado, Ph.D.
Senior Software Engineer
Real-Time Innovations

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