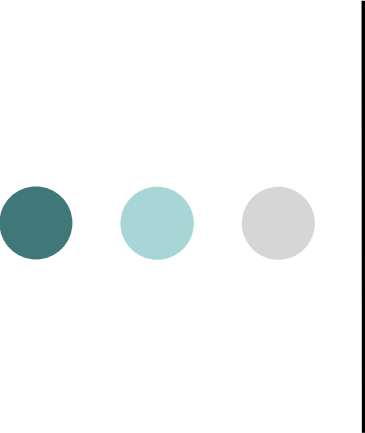




Services Foundation

Msc in Management - Services Science

Giovanna Di Marzo Serugendo

Giovanna.Dimarzo@unige.ch, room B 235, 022 379 00 72

University of Geneva

<http://cui.unige.ch/~dimarzo>



Lecture 2: Summary

Software vs Services

Specific Services

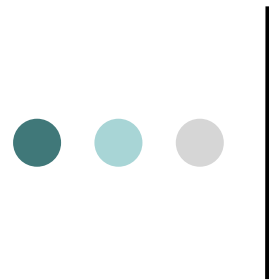
- Web Services
- Context-Aware Services
 - Location-Based Services
- Wearable Computers
- On-Line Games
- Social Media
- E-Government Services
- Services for the elderly
- Smart Systems



Lecture 3 / 4 / 5

Technologies for Services

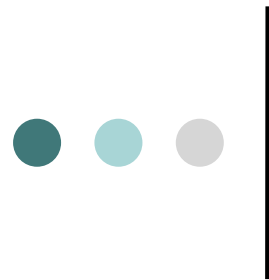
- Interaction Modes (Publish /Subscribe)
- SOA / Mashups / Clouds
- Services Composition and Orchestration



Motivation

Large-scale dynamic distributed applications

- Point-to-point / synchronous communication is impossible
 - Static, rigid applications
- **Loosely coupled** form of interaction
 - Provided by appropriate middleware



Publish/Subscribe

Well adapted to the loosely coupled nature of distributed interaction in large-scale applications (..) with systems based on the publish/subscribe interaction scheme, **subscribers register their interest in an event**, or a pattern of events, and are **subsequently asynchronously notified of events** that match their interest generated by publishers.

[Eugster 03]



Basic Interaction Scheme

Service: event notification service providing storage and management for subscriptions and efficient delivery of events

- Producers **publish information** to an event notification service (an event manager) -- publishers
- Consumers **subscribe** to the information they want to receive from the event notification service -- subscribers
- This information is typically denoted by the term **event**
- The act of delivering it by the term **notification**
- Twitter

[Eugster 03]



Basic Interaction Scheme

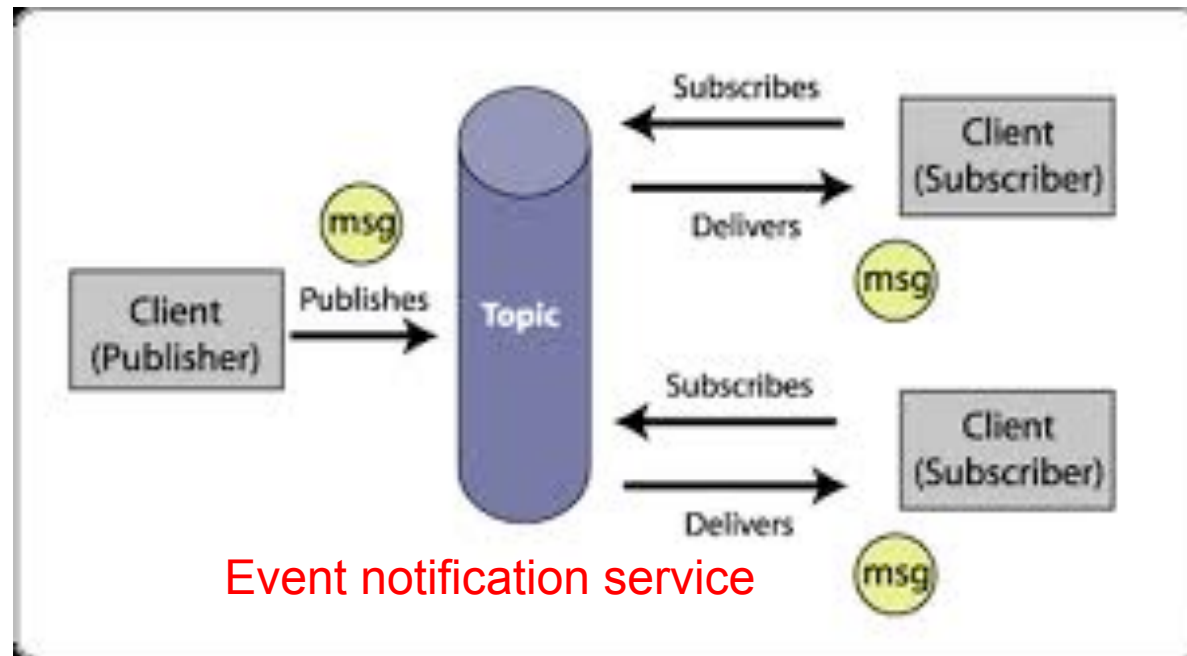
Registration of interests:

- `subscribe()` / `unsubscribe()`
- Event notification service does not forward the subscription to publishers
- Subscribers do not know source of event

Publication of events:

- `publish ()`
- Event notification service propagates event to all subscribers
- **Every** subscriber will be notified of **every** event conforming to its interest

Basic Interaction Scheme





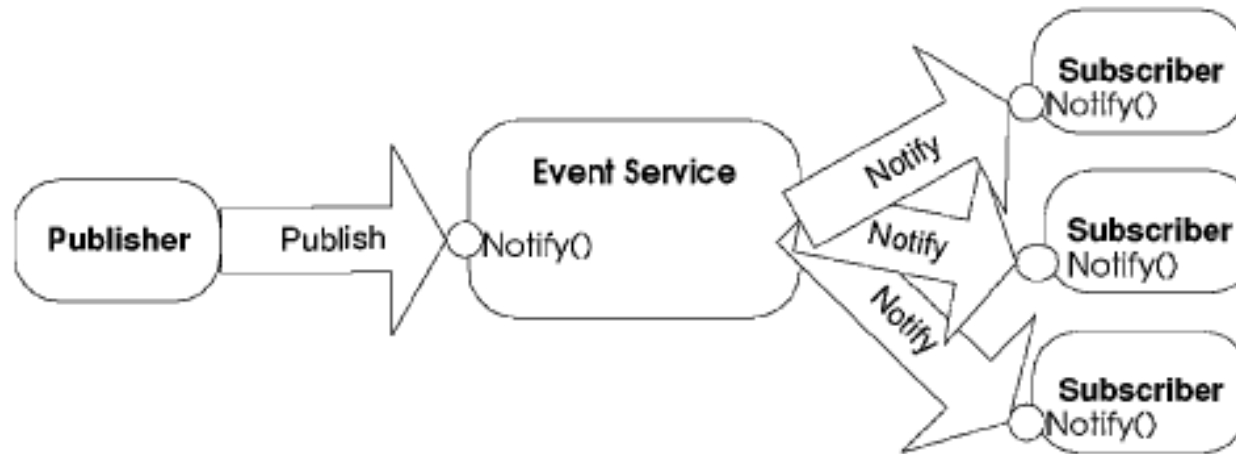
Publish/Subscribe

Decoupling:

- Time
- Space
- Synchronization

between publishers and subscribers

Basic Interaction Scheme

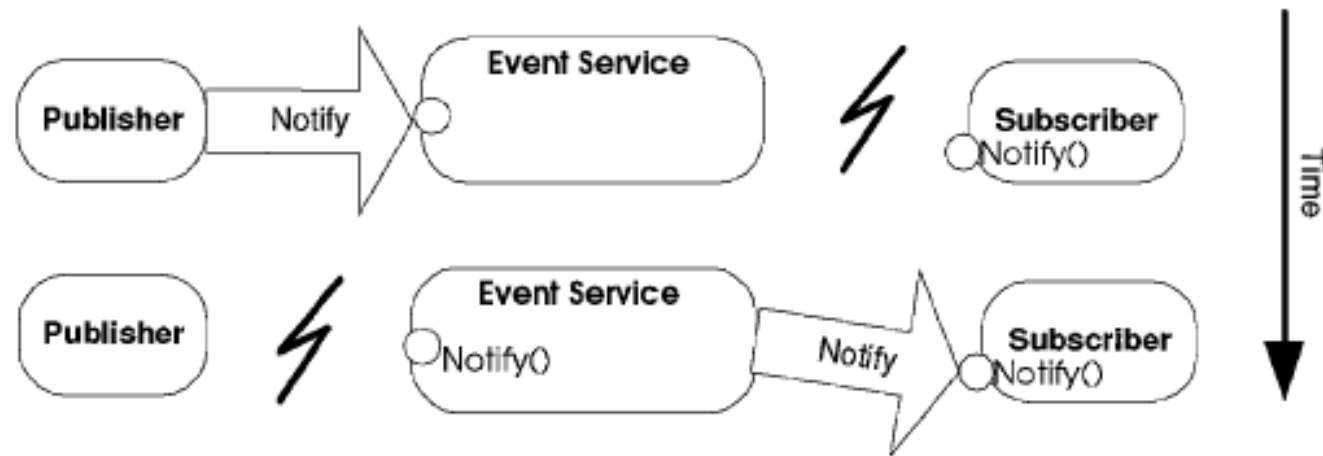


Space decoupling

- Interacting parties **do not know each other**
- Events obtained **indirectly** through the event service
- Publishers **do not hold references** of subscribers and vice-versa
- Subscribers **do not know how many** publishers are involved

[Eugster 03]

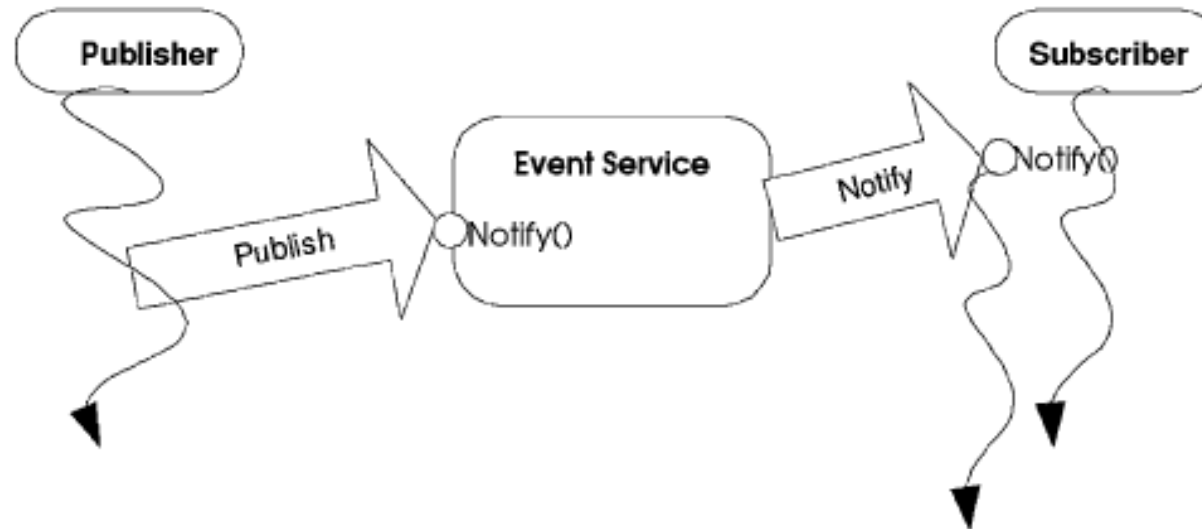
Basic Interaction Scheme



Time decoupling

- Interacting parties **do not need** to be actively **participating** in the interaction **at the same time**
- Publishers might publish events while subscribers are **disconnected**
- Subscribers might get notifications while publishers are **disconnected**
- Skype is **not** Time decoupled (cf chats)

Basic Interaction Scheme



Synchronization decoupling

- Publishers **are not blocked** while producing events
- Subscribers **can get asynchronously notified** of the occurrence of an event while performing some other concurrent activity.
- **Production and consumption** of events do not happen in the main flow of control of the publishers and subscribers, and **do not** therefore **happen in a synchronous manner**.

[Eugster 03]



Basic Interaction Scheme

Advantages of decoupling

- No explicit dependencies
- Increased scalability



Basic Interaction Scheme

Data delivery mechanisms:

- Push vs Pull
- Aperiodic vs Periodic
 - Event based vs arranged schedule
- Unicast (1 to 1) vs 1-to-N (1 to many)



Interactions

Push mechanism

- **Producers periodically push** updated context information to the event service.
 - The event service **maintains** the information in an event store
 - E.g. instant messaging, alerts, sms from operators (to be confirmed)
- ⇒ Better performance, but cost of storing and updates

Pull mechanism

- **Subscriber service must explicitly request** event information.
 - It makes this request on a periodic basis (polling) or when an application demand arises.
 - e.g. emails, rss feeds, twitter
- ⇒ Fewer resources used, but network delays



Interactions

Push





Publish/Subscribe Variants

Subscribers interested in **some** events, **not all** events

Variants

- Topic-based
- Content-based
- Type-based
- Location-based



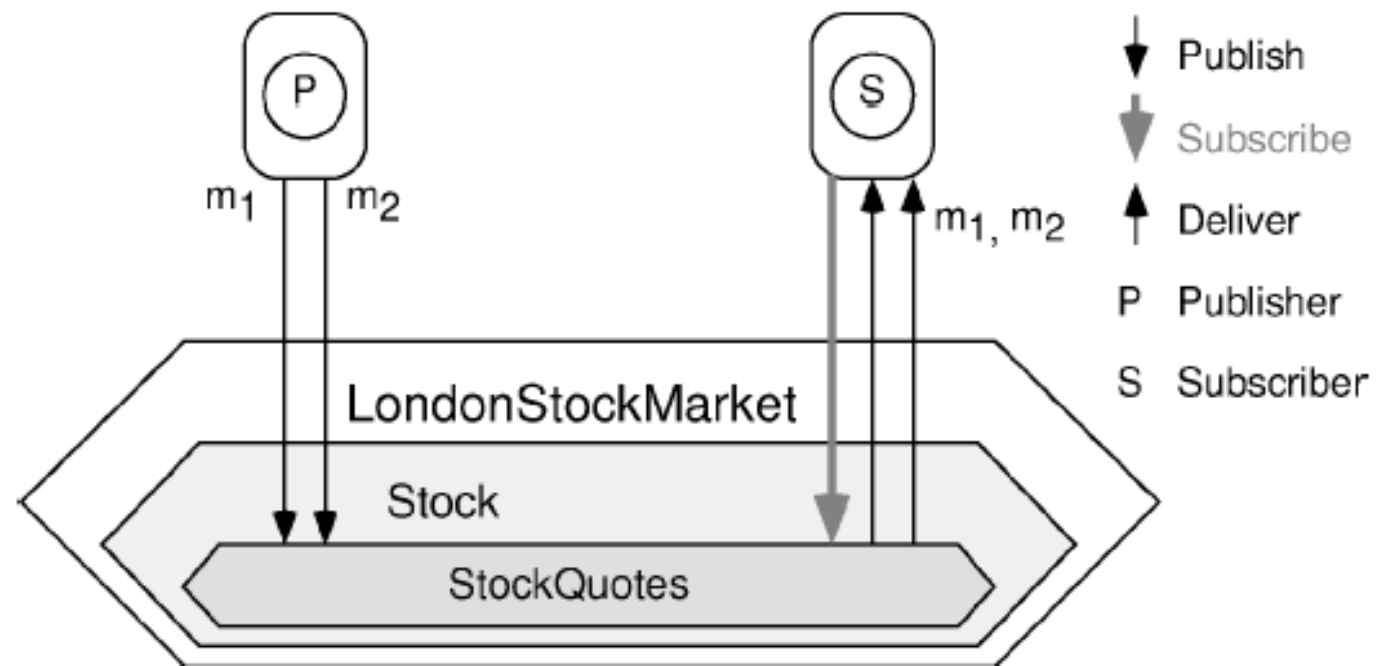
Topic-Based Publish/Subscribe

Topics or Subjects

- Publish events or subscribe to **static** topics using **keywords**
- Maps individual topics to distinct communication channels
 - **Every topic is viewed as an event service** of its own, identified by a unique name
 - Every topic provides an interface offering `publish()` and `subscribe()` operations.



Topic-Based Publish/Subscribe





Topic-Based Publish/Subscribe

```
public class StockQuote implements Serializable {  
    public String id, company, trader;  
    public float price;  
    public int amount;  
}  
public class StockQuoteSubscriber implements Subscriber {  
    public void notify(Object o) {  
        if (((StockQuote)o).company == 'TELCO' && ((StockQuote)o).price < 100)  
            buy();  
    }  
}  
// ...  
Topic quotes = EventService.connect("/LondonStockMarket/Stock/StockQuotes");  
Subscriber sub = new StockQuoteSubscriber();  
quotes.subscribe(sub);
```



Content-Based Publish/Subscribe

Content or property

- Subscription scheme based on the **actual content** of the considered **dynamic** events
 - Events are not classified according to some predefined external criterion (static topic name)
 - Events are **classified according to the properties** of the events themselves
- Consumers subscribe to selective events by **specifying filters** using a subscription language. The filters define constraints, usually in the form of **key-value pairs** of properties and basic comparison operators (and, or, >, <, =)

Content-Based Publish/Subscribe



$m_1: \{ \dots, \text{company: "Telco", price: 120, } \dots, \dots \}$

$m_2: \{ \dots, \text{company: "Telco", price: 90, } \dots, \dots \}$



Content-Based Publish/Subscribe

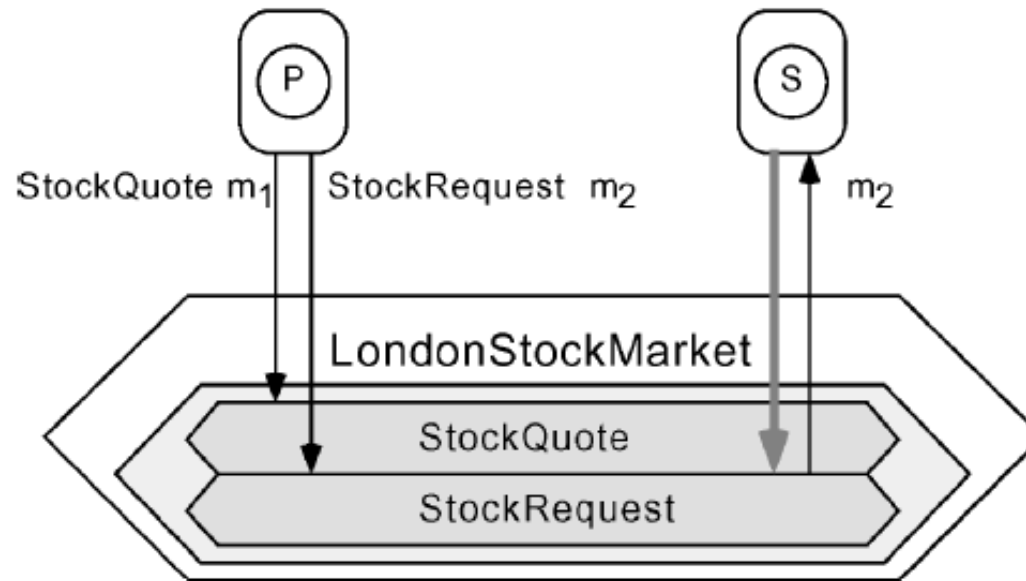
```
public class StockQuote implements Serializable {  
    public String id, company, trader;  
    public float price;  
    public int amount;  
}  
public class StockQuoteSubscriber implements Subscriber {  
    public void notify(Object o) {  
        buy();    // company == 'TELCO' and price < 100  
    }  
}  
// ...  
String criteria = ("company == 'TELCO' and price < 100");  
Subscriber sub = new StockQuoteSubscriber();  
EventService.subscribe(sub, criteria);
```



Type-Based Publish/Subscribe

Event kind or event type

- Subscription scheme based on the **type** of the considered events





Type-Based Publish/Subscribe

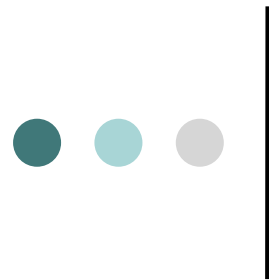
```
public class LondonStockMarket implements Serializable {
    public String getId() {...}
}
public class Stock extends LondonStockMarket {
    public String getCompany() {...}
    public String getTrader() {...}
    public int getAmount() {...}
}
public class StockQuote extends Stock {
    public float getPrice() {...}
}
public class StockRequest extends Stock {
    public float getMinPrice() {...}
    public float getMaxPrice() {...}
}
public class StockSubscriber implements Subscriber<StockQuote> {
    public void notify(StockQuote s) {
        if (s.getCompany() == 'TELCO' && s.getPrice() < 100)
            buy();
    }
}
// ...
Subscriber<StockQuote> sub = new StockSubscriber();
EventService.subscribe<StockQuote>(sub);
```



Location-Based Publish/Subscribe

Mobile applications

- Anonymous communication based on location
- Topic = location criteria
 - "I subscribe to all events published by peers **located within a given range**"
- Topic and content-based publish /subscribe
 - Event is published in a particular *geographical context*
 - Matching process is performed dynamically on this context, and on the content of events.
 - "I subscribe **to all traffic jam events** published by **peers located within a given range**"



Events Notifications

Messages

- Structure (header, body)
- Encoded in XML (for instance)
- Sent to subscribers through `notify()`

Invocations

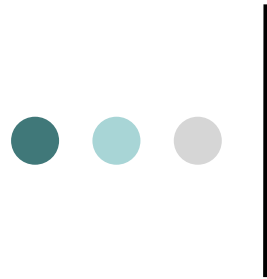
- Triggers execution of specific operation (e.g. method call) at subscriber side



Publish/Subscribe Alternatives

Alternatives

- Message Passing
- Remote Invocation
- Notifications
- Shared Spaces
- Message Queuing



Alternatives

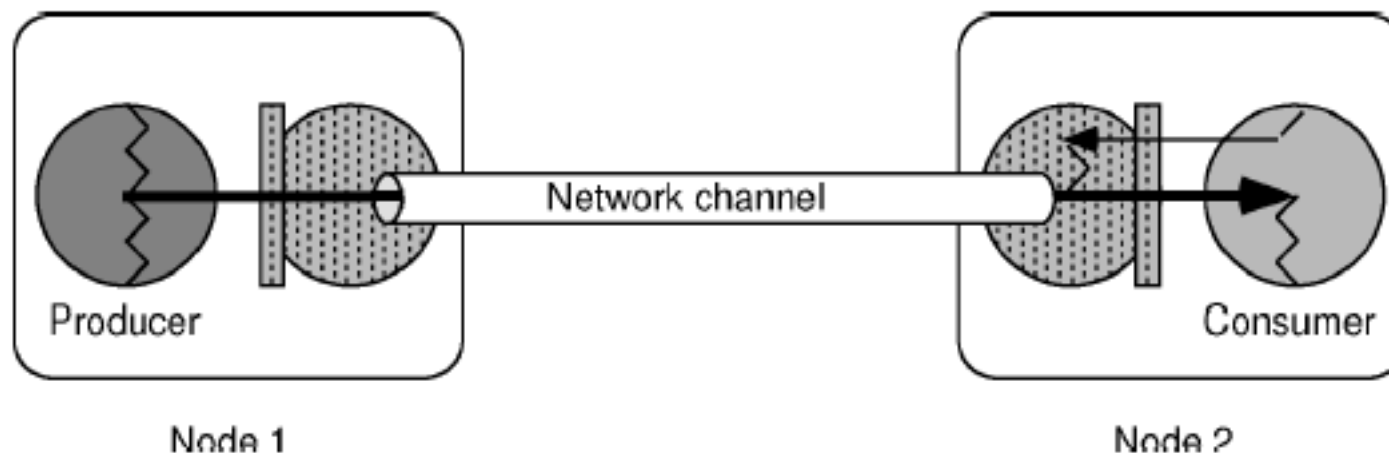
Message Passing

- Participants communicate by sending and receiving messages
- Message passing is asynchronous for the producer
- Message consumption is generally synchronous
- Producer and consumer **are coupled both in time and space**
- Producer and consumer must both be active at the same time
- Recipient of a message is known to the sender
- Skype ?



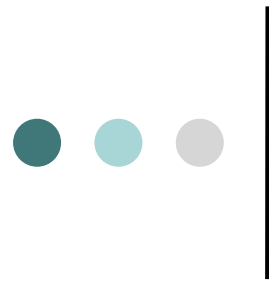
Alternatives

Message Passing



- Producer **sends messages asynchronously** through a **communication channel** (previously set up for that purpose).
- Consumer **receives messages by listening synchronously** on that channel

[Eugster 03]



Alternatives

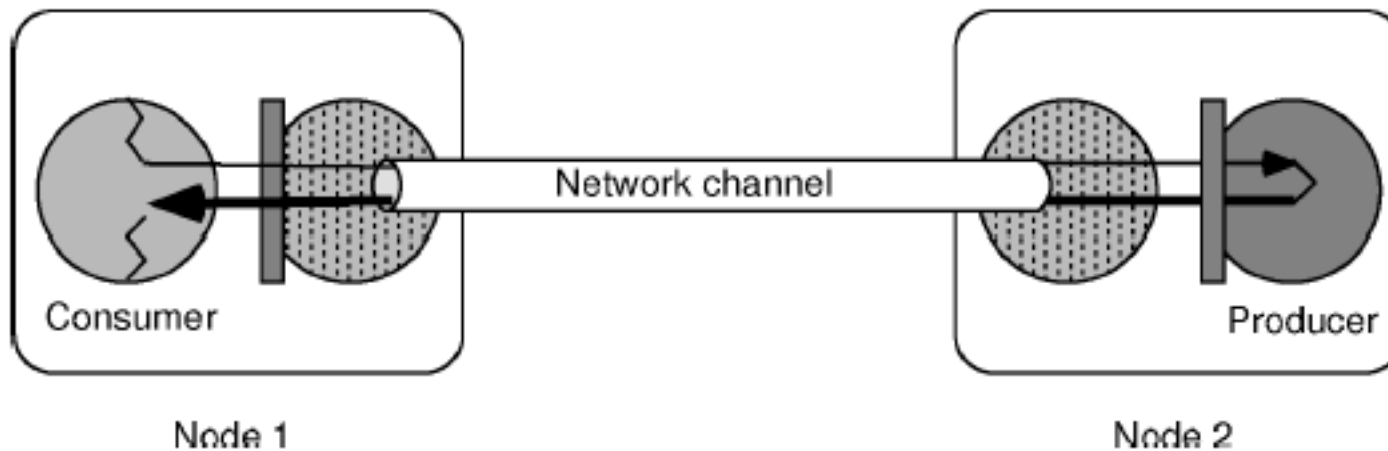
Remote Invocation (RPC)

- Remote interactions appear the same way as local interactions
- Remote invocation is synchronous for producer
- Remote invocation is asynchronous for consumer
- Producers and consumers are **coupled in time and space**
- Consumer holds a reference to producer – to return a value



Alternatives

Remote Invocation (RPC)



- Producer performs a **synchronous call** (calls and waits for a reply)
- Consumer processes the call **asynchronously** (provides a reply)

[Eugster 03]



Alternatives

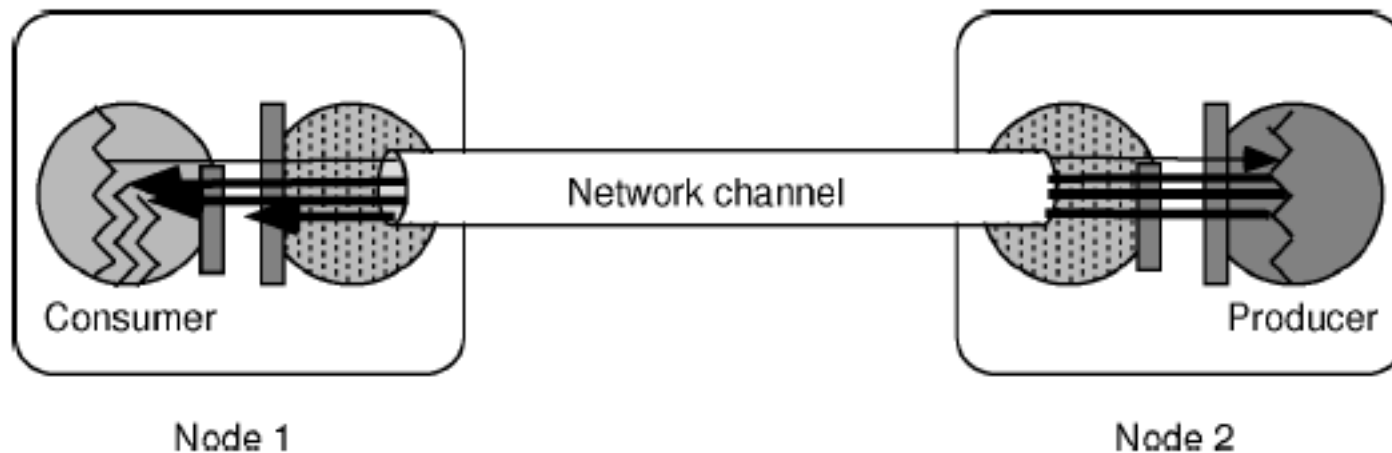
Notifications

- Synchronous remote invocation is split into two asynchronous invocations
 - the first one sent by the client to the server
 - the second one sent by the server to the client to return the reply
- Subscribers register their interest directly with publishers
- Publishers manage subscriptions and send events



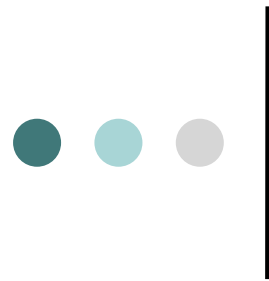
Alternatives

Notifications



Producers and consumers communicate using **asynchronous invocations** in both directions

[Eugster 03]



Alternatives

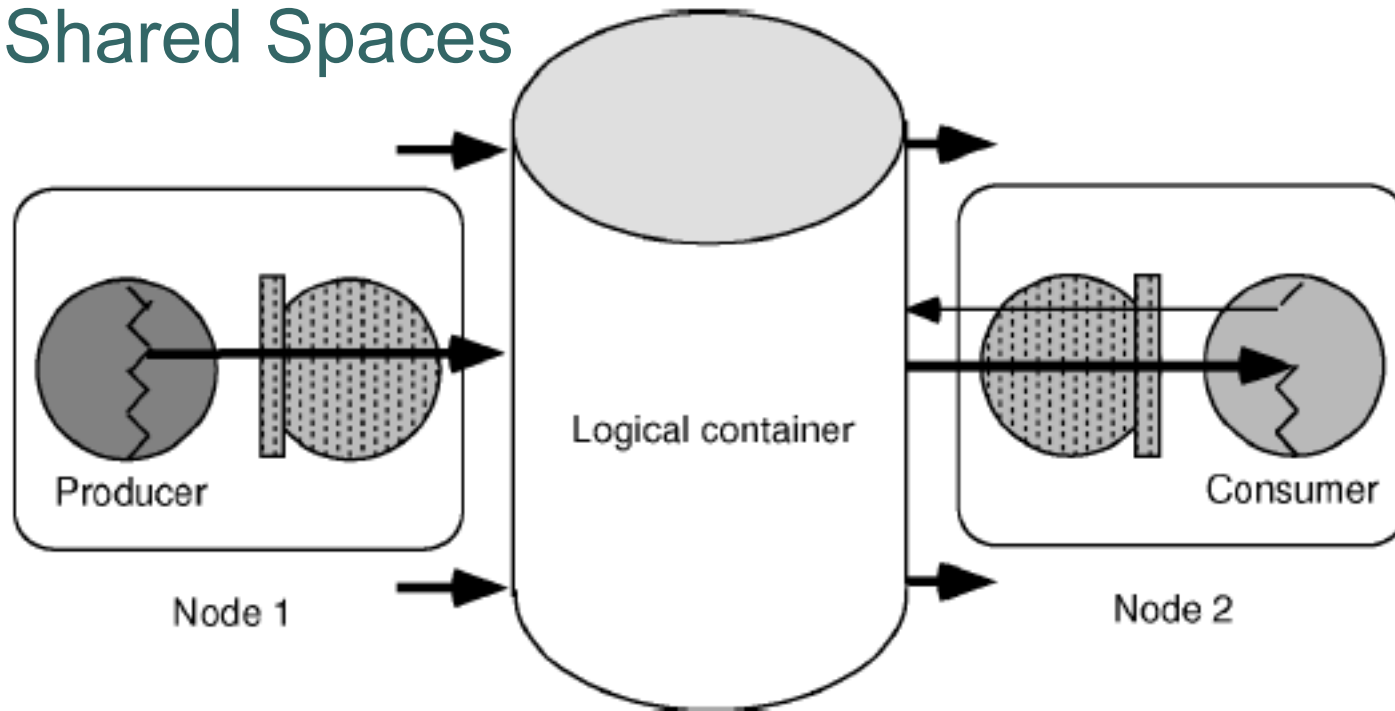
Shared Spaces

- **Shared** memory spaces
 - Tuple spaces, Linda [Gelernter 95]
- **Time and space decoupling**
- Publishers insert information into space (no knowledge of further use)
 - `out ( )`
- Consumer pulls information from space in a **synchronous** manner (pattern matching)
 - `in( )`, `read( )`



Alternatives

Shared Spaces



Producers **insert data asynchronously** into the shared space
Consumers **read data synchronously**

[Eugster 03]



Alternatives

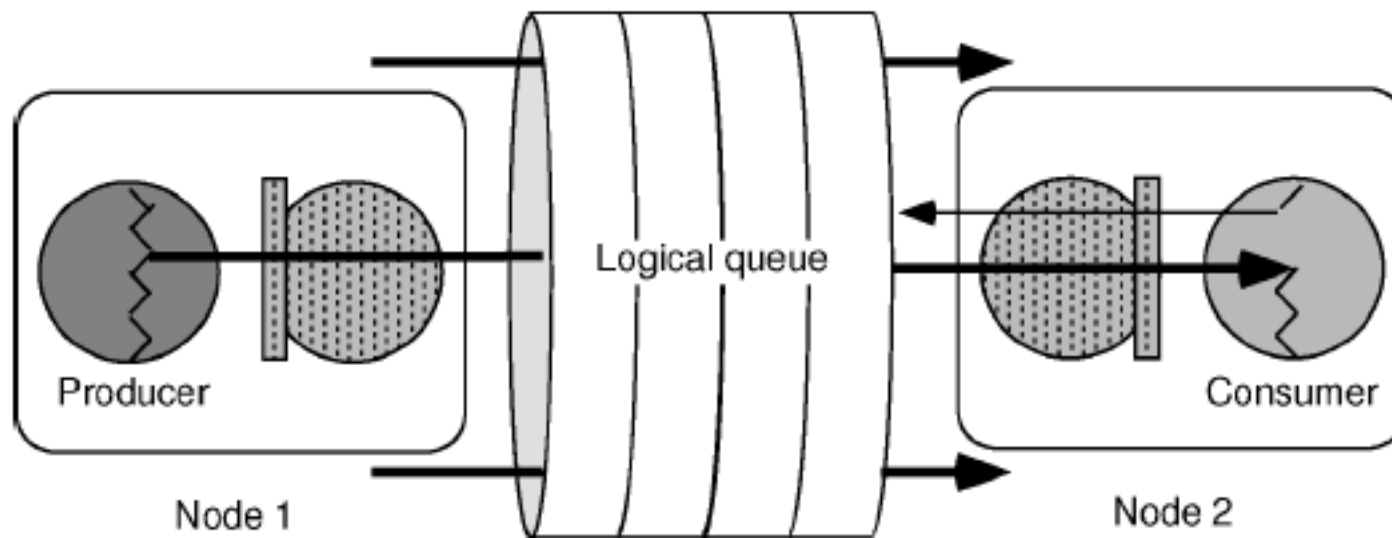
Message Queuing

- Message-oriented middleware
- Producers and consumers are **decoupled in both time and space**.
- Consumers **synchronously pull messages**
- Transactional, timing, and ordering guarantees



Alternatives

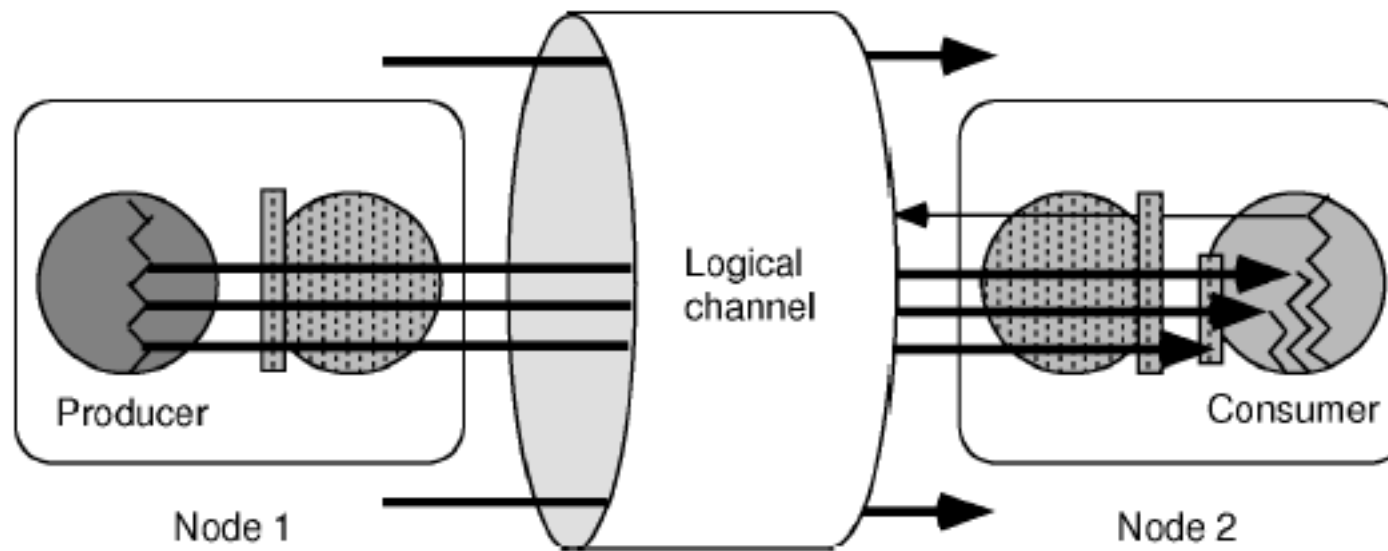
Message Queuing



- Messages are stored in a **FIFO queue**.
- Producers **append messages asynchronously** at the **end** of the queue
- Consumers **dequeue** them **synchronously** at the **front** of the queue.

[Eugster 03]

● ● ● | Publish/Subscribe



Producers and consumers are **decoupled** in terms of **space**, **time**, and **synchronization**.

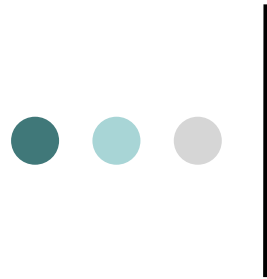
[Eugster 03]



Decoupling - Alternatives

Abstraction	Space decoupling	Time decoupling	Synchronization decoupling
Message passing	No	No	Producer-side
RPC/RMI	No	No	Producer-side
Asynchronous RPC/RMI	No	No	Yes
Future RPC/RMI	No	No	Yes
Notifications (observer pattern)	No	No	Yes
Tuple spaces	Yes	Yes	Producer-side
Message queuing (Pull)	Yes	Yes	Producer-side
Publish/subscribe	Yes	Yes	Yes

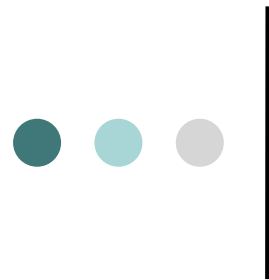
[Eugster 03]



Summary

Interaction Modes

- Publish / Subscribe
 - Basic interaction scheme
 - Data delivery mechanism
 - Push / Pull
 - Topic- / Content- / Type- based
 - Location-based publish/subscribe
- Alternatives
 - Message passing / RPC / Shared spaces / Notifications / Message queuing



Recommended Reading

[Eugster 03] P. Eugster, P. Felber, R. Guerraoui, A. Kermarrer. The many faces of publish/subscribe. ACM Computing Surveys, Vol. 35, No. 2, June 2003, pp. 114–131.

http://infoscience.epfl.ch/record/52371/files/IC_TECH_REPORT_200104.pdf

[Eugster 05] P. Eugster, B. Garbinato, Adrian Holzer: Location-based Publish/Subscribe. NCA 2005: 279-282

[Gelernter 85] Gelernter, D. 1985. Generative communication in Linda. ACM Trans. Program. Lang. Syst. 7, 80–112.