Technische Universität Darmstadt

Telecooperation

Ubiquitous & Mobile Computing

Connectivity: Distributed Event-based Systems (DEBS)

Dr. Erwin Aitenbichler

Outline

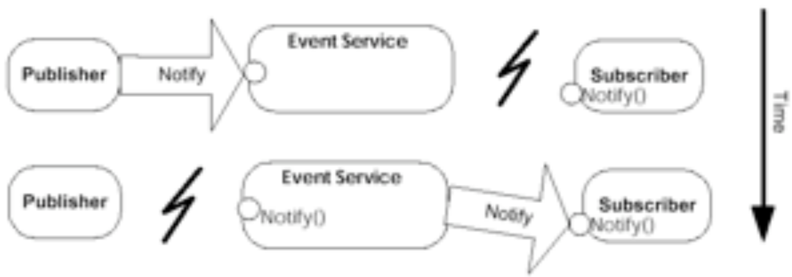
- Interaction models in distributed systems
  - Callbacks
  - Message Queues
  - Publish/Subscribe
- Publish/Subscribe Systems
  - Classification
  - Adressing: Channel-based, Subject-based, Content-based, Type-based, and Concept-based
  - Subscription Mechanisms
  - Distributed Event Systems
  - Data and Filter Models
  - Filter covering, overlapping, and merging
  - Routing
  - Advanced Functions: QoS, Transactions, and Mobility Support
  - Example Systems

Dr. Erwin Aitenbichler
Prof. Dr. M. Mühlhäuser
Telekooperation
©

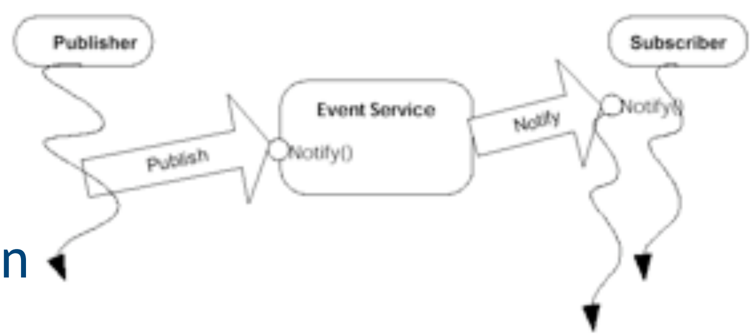
Towards loosely coupled systems

1. Space decoupling
   - parties don‘t know each other
   - 1-to-many comm. possible

2. Time decoupling:
   - parties not (necessarily) active at same time



3. Flow decoupling
   - event production & consumption ∉ main control flow



- 1, 2, 3: coordination & synchronization drastically reduced

Interaction Models

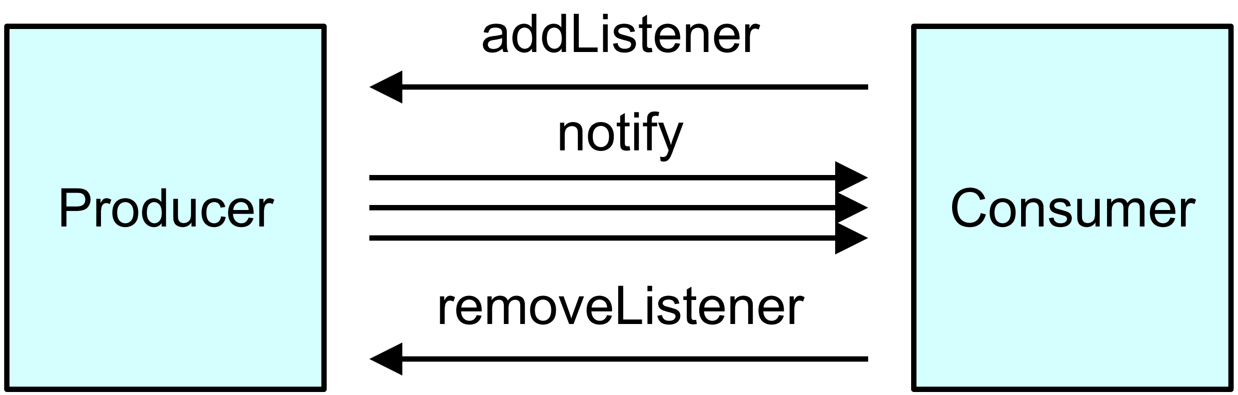
- Interaction Models in Distributed Systems can be classified according to
  - who **initiated** the interaction
  - how the communication partner is **addressed**

	Consumer-initiated („pull“)	Provider-initiated („push“)
Direct Addressing	Request/Reply	Callback
Indirect Addressing	Anonymous Request/Reply	Event-based

- Provider: provides data or functionality
- Anonymous Request/Reply: provider is selected by communication system and not specified directly (e.g., IP Anycast)

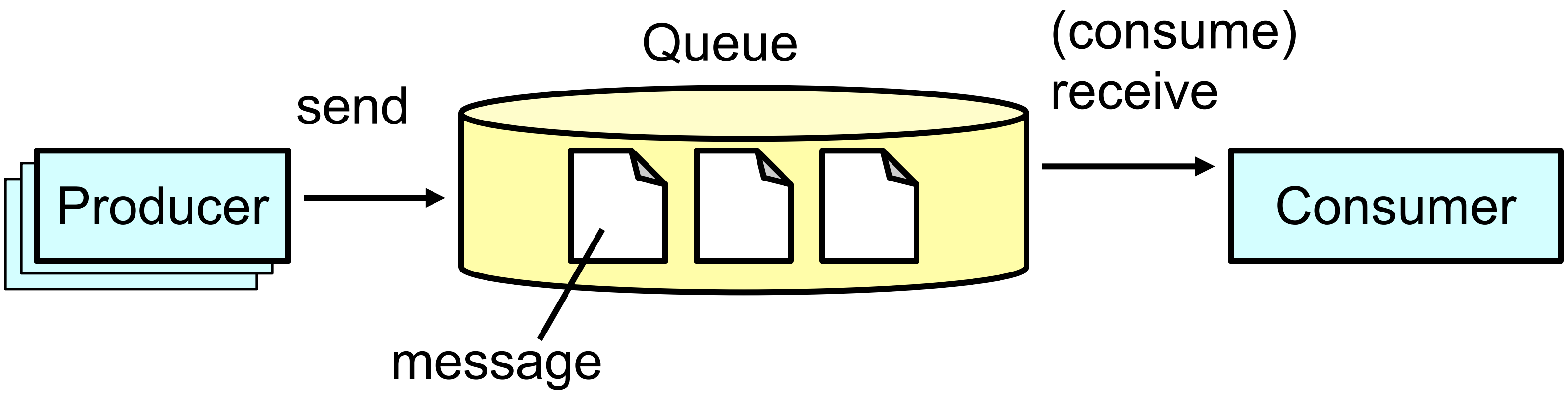
Concepts: Callbacks

- Synchronous (remote) method calls often used to emulate behavior of event-based systems
  - See also: Observer Design Pattern
  - Frequently used in GUI toolkits; example:



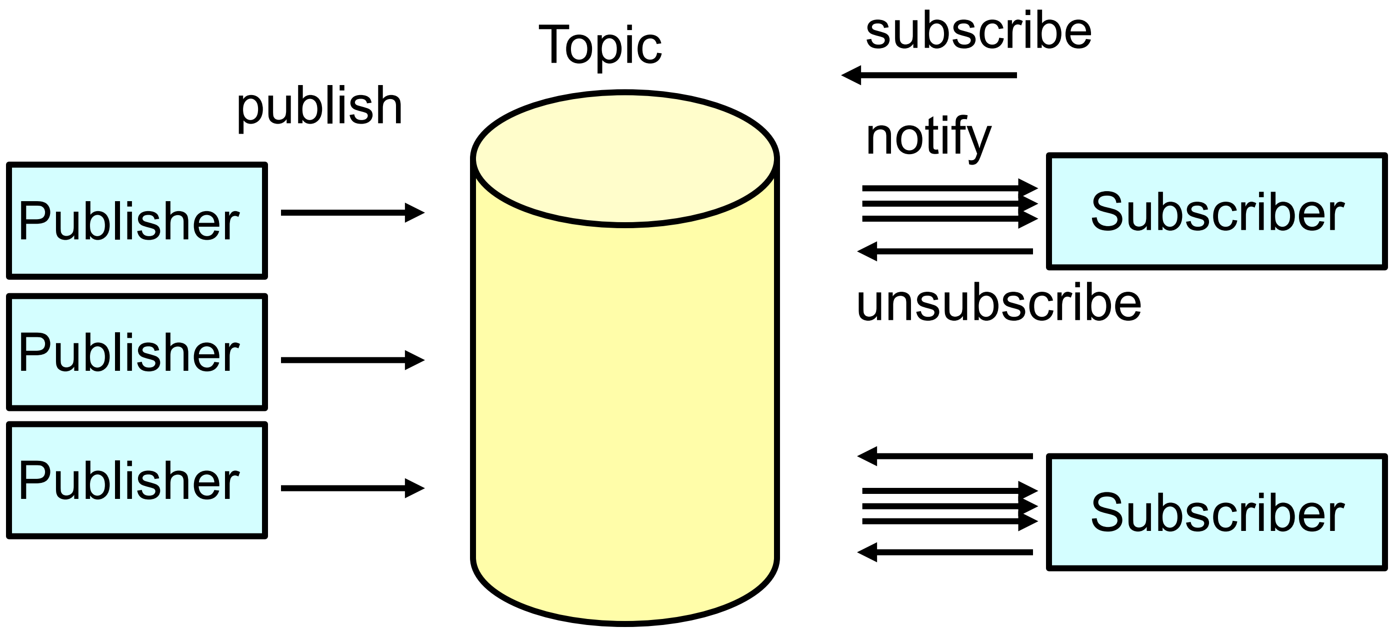
  - P & C coupled in space and time, decoupled in flow
  - Producers have to take care of subscription management and error handling

Concepts: Message Queues



- Each message has only one consumer
- Receiver acknowledges successful processing of message
- No timing dependencies between sender and receiver
- Queue stores message (persistently), until
  - It is read by a consumer
  - The message expires (Leases)

Concepts: Publish/Subscribe



- Here: Topic-based Publish/Subscribe
  - Interested parties can subscribe to a topic (channel)
  - Applications post messages explicitly to specific topics
- Each message may have multiple receivers
- Full decoupling in space, time, and flow

Terms

- **Event:** Any happening in the real world or any kind of state change inside an information system that is observable
- **Notification:** The reification of an event as a data structure
- **Message:** Transport container for notifications and control messages

Classification (1)

- Messaging Domain
  - Point-to-Point (Producer -> Consumer)
  - Subscription-based Pub/Sub
  - Advertisement-based Pub/Sub
- Subscription Mechanism
  - Channel-based (=Topic-based) Subscription
  - Content-based Subscription
  - Subject-based Subscription (limited form of Content-based sub.)
- Server Topology
  - Single Server (Elvin3)
  - Hierarchical (TIB/Rendezvous, JEDI, Keryx)
  - Acyclic Peer-to-Peer
  - Generic Peer-to-Peer (SIENA)
  - Hybrid

Classification (2)

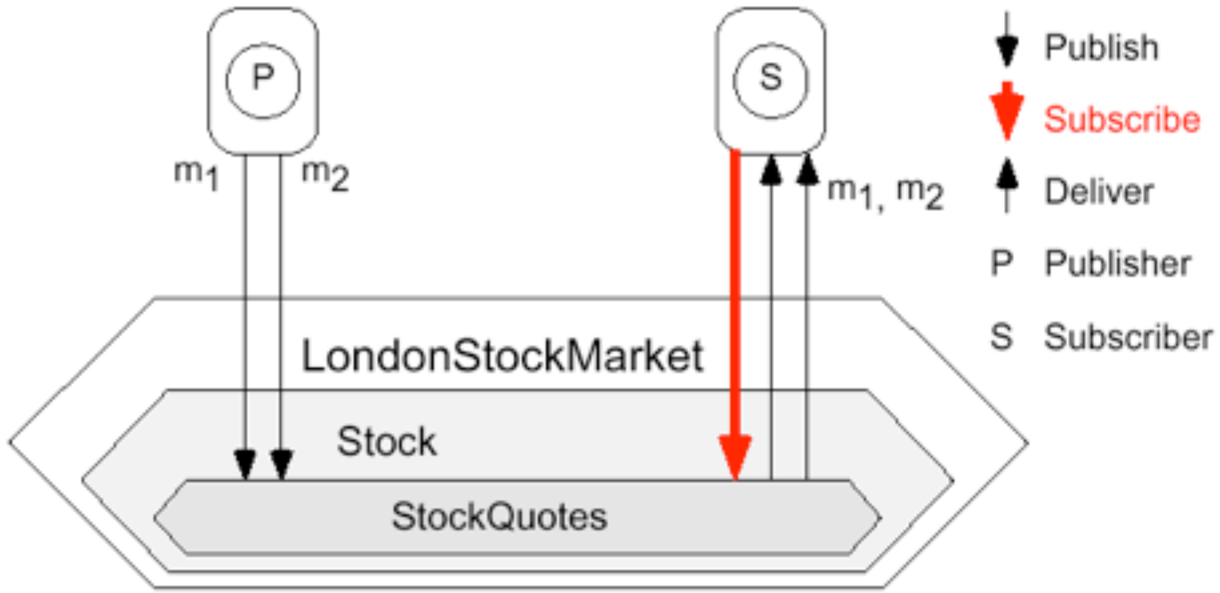
- Event Data Model
  - Untyped
  - Typed
  - Object-oriented
- Event Filters
  - Expressiveness and flexibility of subscription language
    - Simple Expressions
    - SQL-like Query Language
    - (Mobile) Code
  - Pattern Monitoring: Temporal sequence of events
  - Evaluated in router network
- Note: Scalability ↔ Expressiveness Tradeoff
  - Simple Expressions permit Filter Merging → better scalability

Classification (3)

- Features
  - Scalability
  - Security
  - Client Mobility
    - Transparent
    - Native
    - External
  - Disconnection
  - QoS
    - Reliability
    - Response Time (Real-Time Constraints)
  - Transactions
  - Exception Handling

Addressing

- Channel-based Addressing (=Topic-based)
  - Interested parties can subscribe to a channel
  - Application posts messages explicitly to a specific channel
  - Channel Identifier is only part of message visible to event service
  - There is no interplay between two different channels



Addressing

- Channel-based Addressing (=Topic-based)

  Extension: Topic Hierarchies (SwiftMQ)

  <roottopic>.<subtopic>.<subsubtopic>

  - Messages are published to addressed node and all subnodes

    iit.sales -> iit.sales.US, iit.sales.EU

  - Subscribing means receiving messages addressed to this node, all parent nodes and all sub nodes:

    Subscription to iit.sales

    Receives from: iit, iit.sales, iit.sales.US, iit.sales.EU

    But not from: iit.projects

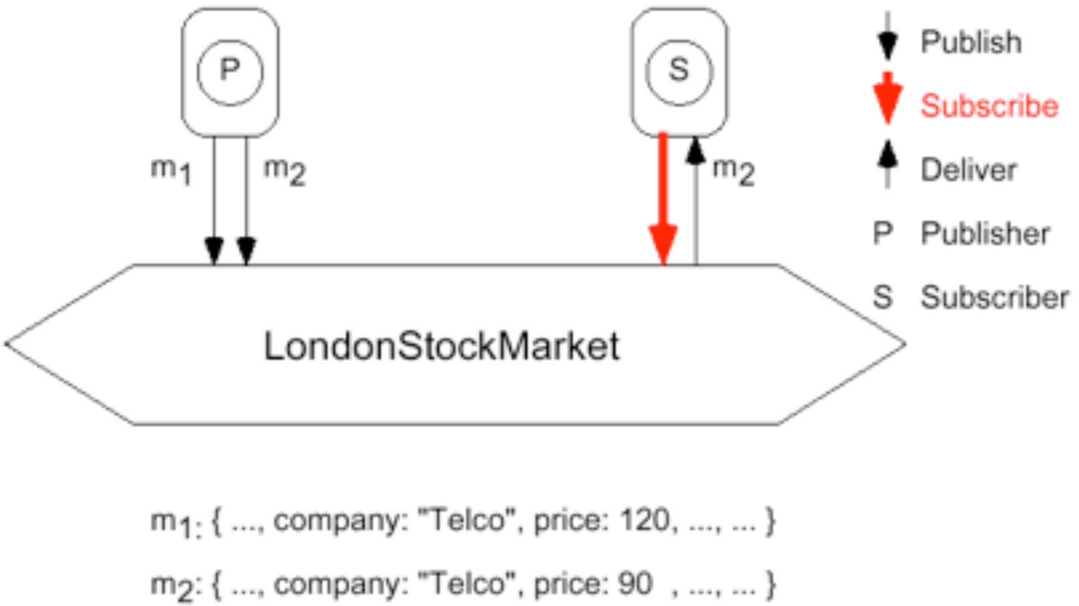
  - Subscriber receives each message only once

Addressing

- Subject-based Addressing
  - Limited form of Content-based Subscription
  - Notifications contain a well-known attribute – the subject – that determines their address
  - Subscriptions express interest in subjects by some form of expressions to be evaluated against the subject
  - Subject is
    - List of strings (TIB/Rendezvous, JEDI)
    - Properties: Typed Key/Value-Pairs (JMS)
  - Subject (= header of notification) is visible to event service, remaining information is opaque
  - Subscription is
    - (Limited form of) Regular Expressions over Strings (TIB, JEDI)
    - (Limited form of) SQL92 Queries (JMS)
  - Filtering is done in the Router Network!

Addressing

- Content-based Subscription
  - Domain of filters extended to the whole content of notification
  - More freedom in encoding data upon which filters can be applied
  - More information for event service to set up routing information



Addressing

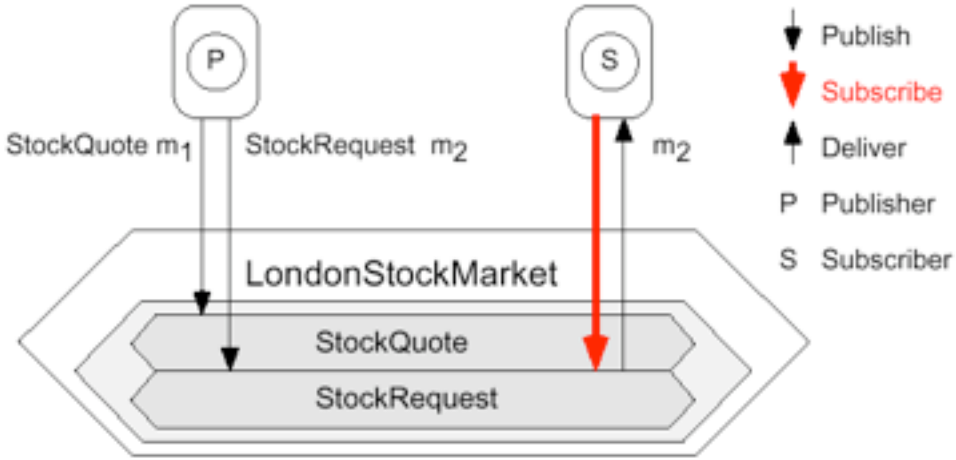
- Content-based Addressing

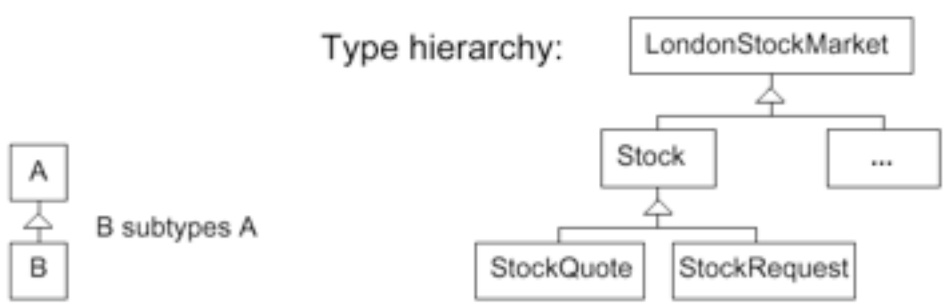
  Content-based <-> Subject-based
  - Subject-based requires some preprocessing by publisher
    - Information that might be used by subscribers for filtering must be placed in header fields
    - Thus producer makes assumptions about subscribers‘ interests
  - Content-based
    - Subscribers exclusively describe their interests in filter expressions

- Concept-based Addressing
  - Provides higher level of abstraction for description of subscribers’ interests
  - Matching of notifications and transformation of notifications based on ontologies

Addressing

- Type-based Addressing
  - Similar to channel-based Pub/Sub with hierarchies
  - Supports subtype tests (instanceof)
  - Good integration of middleware & language, type safety



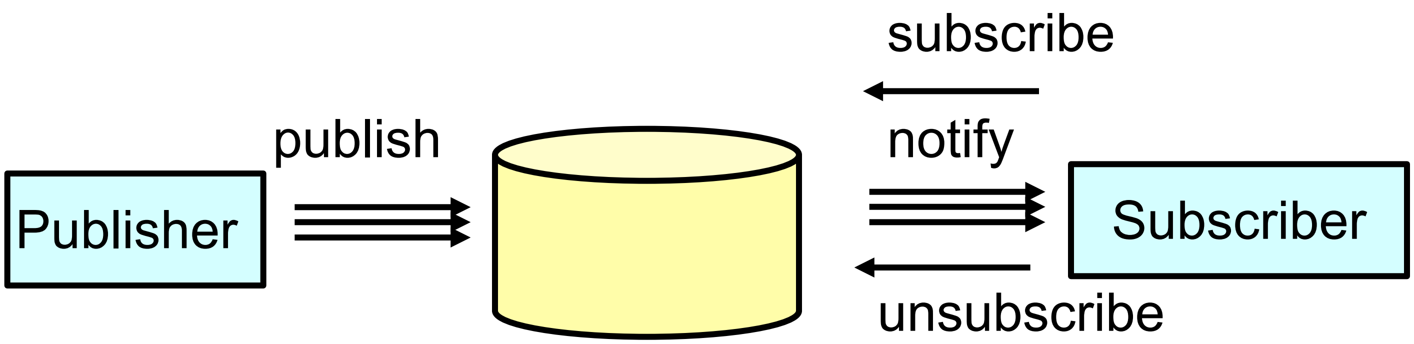


Application Examples

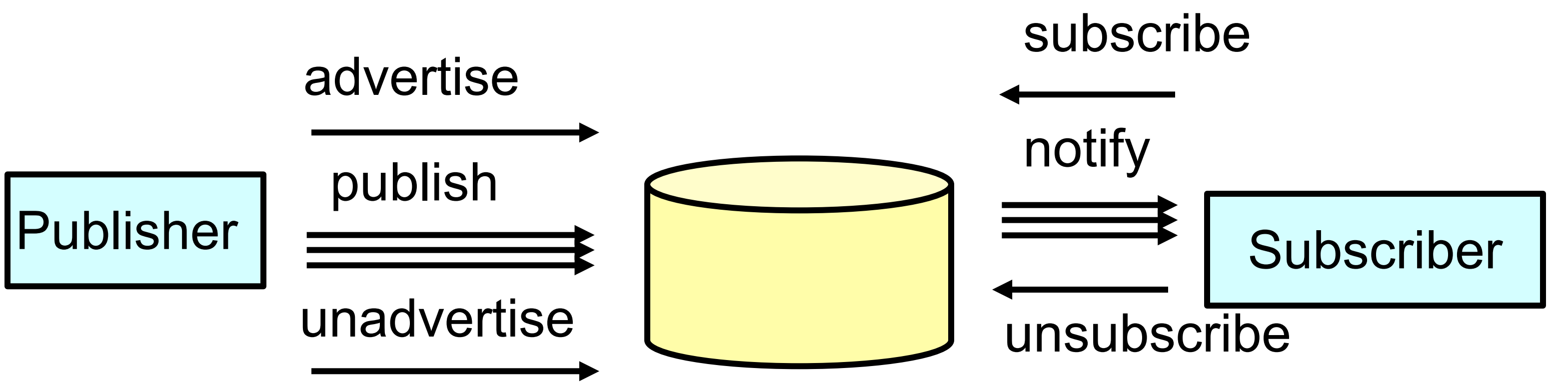
Methods	Applications
Channel-based addressing	Distributed Object Computing (object location and execution transparency) Media streaming
Content-based addressing	Context events (from sensors)
Type-based addressing	Service discovery (interface matching)
Content-based addressing with subscriptions	RFID middleware
Content-based addressing with advertisements	Stock quotes

Subscription Mechanisms

- Subscription-based

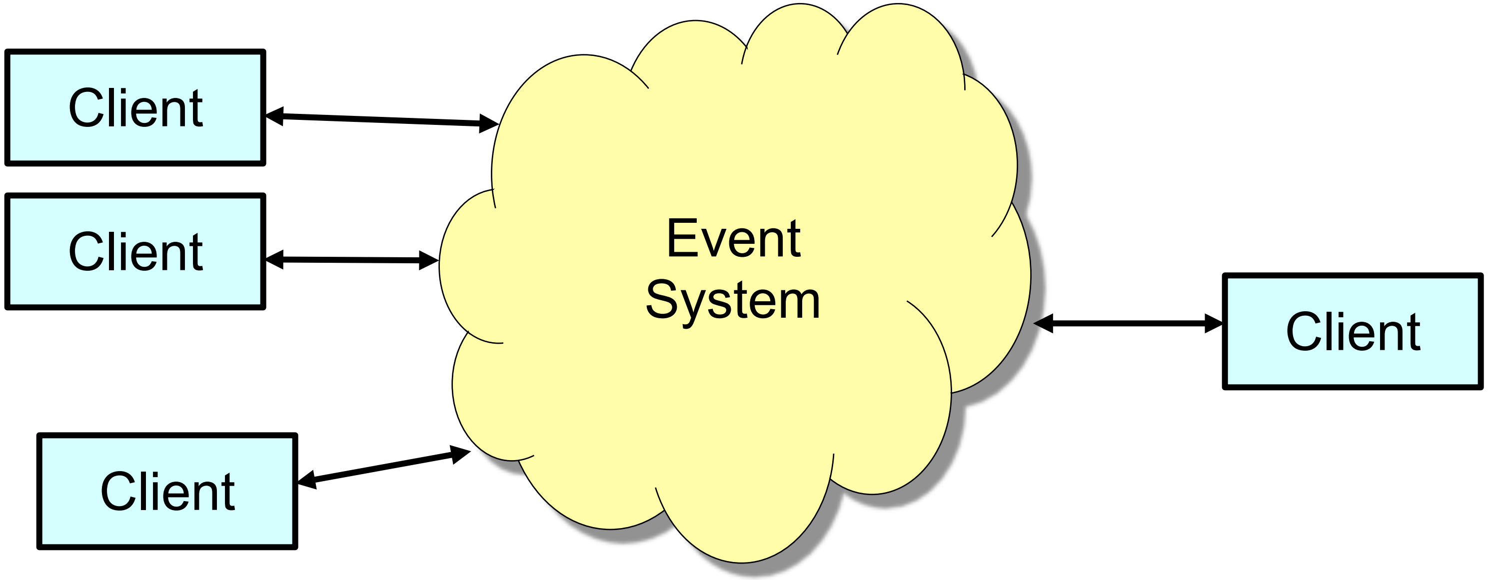


- Advertisement-based



Distributed Event Systems

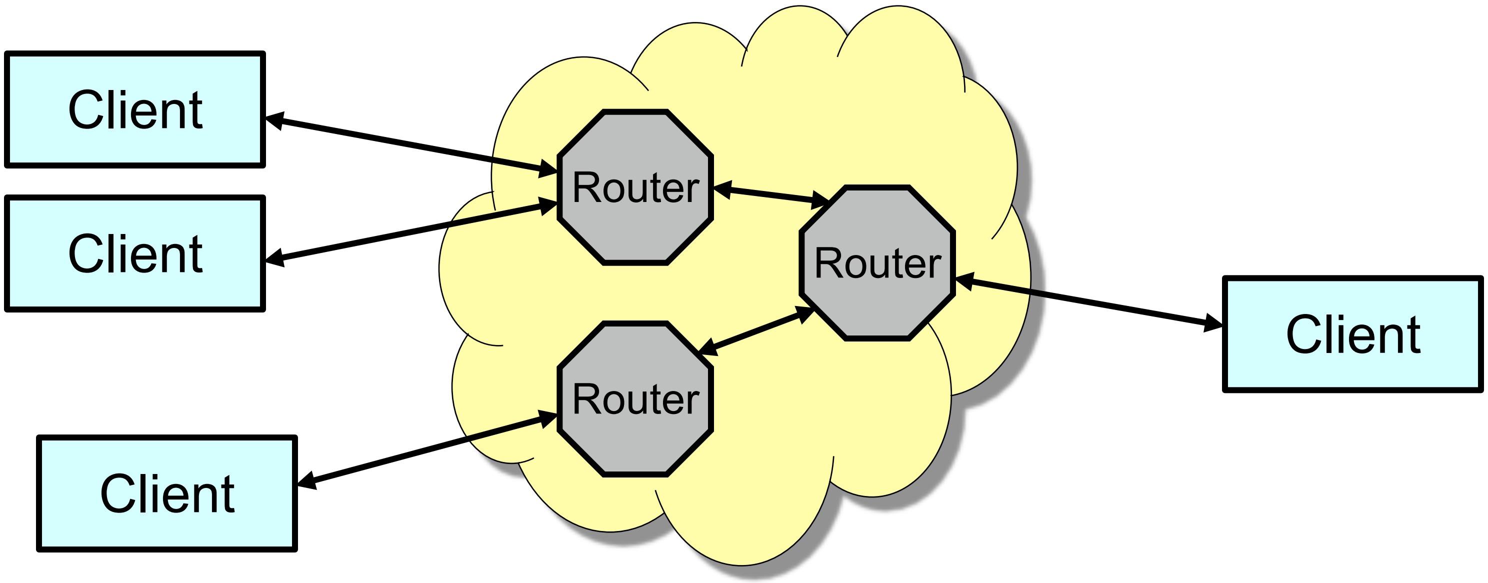
- (Distributed) Event Systems
  - permit loosely coupled, asynchronous point-to-multipoint communication patterns
  - are application independent infrastructures
  - Only clients communicating via a logically centralized component



Distributed Event Systems

- Logically centralized component
  - Single server or Network of event routers
  - Transparent for application (=Client)
    Router network can be reconfigured independently and without changes to the application

  => Scalability



Data Model

- Notification

  consists of a nonempty set of attributes $\{a_1, \ldots, a_n\}$

  An attribute is a triple $a_i = (n_i, t_i, v_i)$, where
  - n_i is the attribute name
  - t_i is the attribute type, and
  - v_i is the value
- All data models can be mapped to this representation
  - Hierarchical messages in which attributes may be nested are flattened by using a dotted naming scheme, e.g.,

    $\{(pos, set, \{(x, int, 1), (y, int, 2)\})\}$ can be rewritten as

    $\{(pos.x, int, 1), (pos.y, int, 2)\}$
  - Objects can be externalized into a tree structure

Attribute Filters

- An *attribute filter* is a simple filter that imposes a constraint on the value and type of a single attribute. It is defined as a tuple

  $$A = (n, t, op, c)$$

  where
  - *n* is the name of the attribute to test
  - *t* is the expected value type,
  - *op* is the test operator, and
  - *c* is a constant that serve as parameter for the operator

- An attribute *a* matches an attribute filter *A*, iff

  $$a \sqsubseteq A :\Leftrightarrow n_A = n_a \wedge t_A = t_a \wedge op_A(v_a, c_A)$$

Filters

- A **filter** is composed of one or more attribute filters. While attribute filters are applied to single attributes, filters are applied to whole notifications

- Filters that only consist of a single attribute filter are called **simple filters**, i.e., $F = \{ A_1 \}$

- Filters containing multiple attribute filters are called **compound filters**, i.e. $F = \{ A_1, \ldots, A_n \}$

- In the following, we only consider compound filters that only use conjunctions - also called conjunctive filters.
- Arbitrary logic expressions can be written as conjunctive filters in one or multiple subscriptions

- A notification *n* **matches** a filter *F*, iff it satisfies all attribute filters of *F*:

$$n \sqsubseteq F :\Leftrightarrow \forall A \in F : \exists a \in n : a \sqsubseteq A$$

Dr. Erwin Aitenbichler
Prof. Dr. M. Mühlhäuser
Telekooperation
©

Matching: Example

Filter		**Message**
String event=alarm	*matches*	String event=alarm Time date=02:40:03
String event=alarm Integer level>3	*not matches*	String event=alarm Time date=02:40:03

Covering

- Covering between attribute filters:
  - An attribute filter A_1 *covers* another attribute filter A_2, iff

$$A_1 \sqsupseteq A_2 :\Leftrightarrow n_1 = n_2 \wedge t_1 = t_2 \wedge L_A(A_1) \supseteq L_A(A_2)$$

  - where L_A is the set of all values that cause an attribute filter to match

$$L_A(A_i) = \{v \mid op_i(v, c_i) = true\}$$

- Covering between filters:
  - A filter F_1 *covers* another filter F_2, iff for each attribute filter in F_1 there exists an attribute filter in F_2 that is covered by the attribute filter in F_1:

$$F_1 \sqsupseteq F_2 :\Leftrightarrow \forall i\, \exists j : A_{1,i} \sqsupseteq A_{2,j}$$

  - The covering relations are required to identify and merge similar filters

Overlapping

- The filters F_1 and F_2 are *overlapping*, iff

$$F_1 \sqcap F_2 :\Leftrightarrow$$
$$\neg\exists A_{1,i}, A_{2,j} : (n_{1,i} = n_{2,j} \wedge (t_{1,i} \neq t_{2,j} \vee L_A(A_{1,i}) \cap L_A(A_{2,j}) = \emptyset))$$

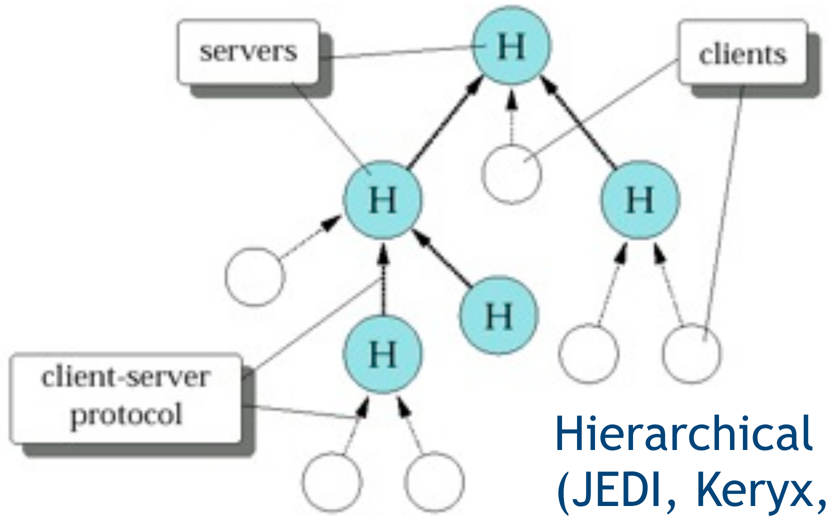
- The overlapping relation is required to implement advertisements.

- When an advertisement A overlaps with a subscription S, we say that A **is relevant** for S.
- As a consequence, all notifications published by the client that issued A must be forwarded to the clients that issued S.

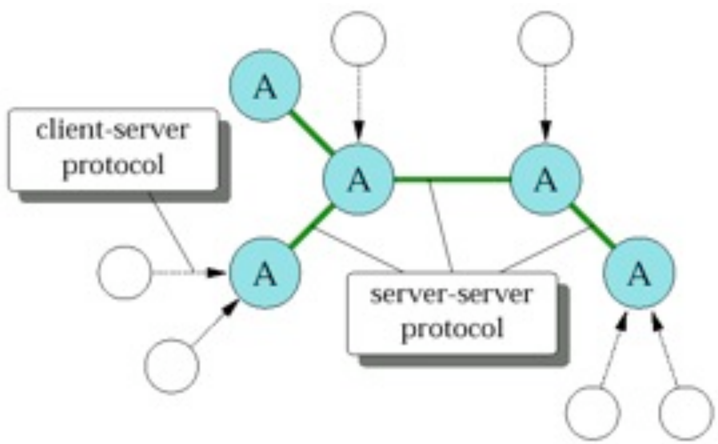
Dr. Erwin Aitenbichler
Prof. Dr. M. Mühlhäuser
Telekooperation
©

Router Topologies

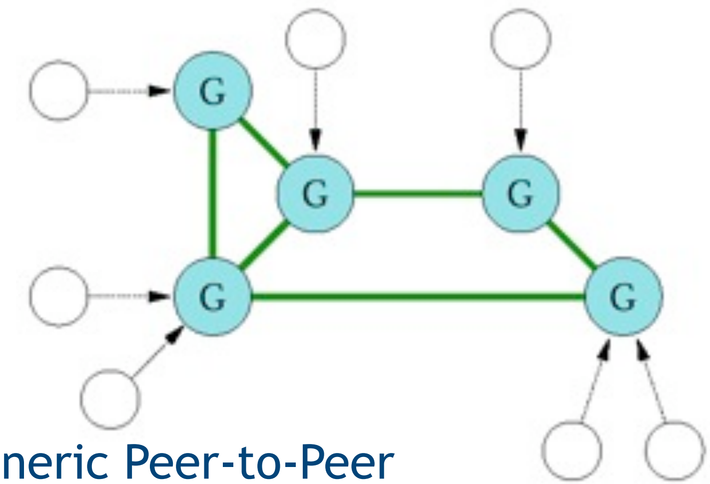
Centralized Server
(Elvin3)



Hierarchical
(JEDI, Keryx, TIB)



Acyclic Peer-to-Peer



Generic Peer-to-Peer
(SIENA)

Routing of Requests

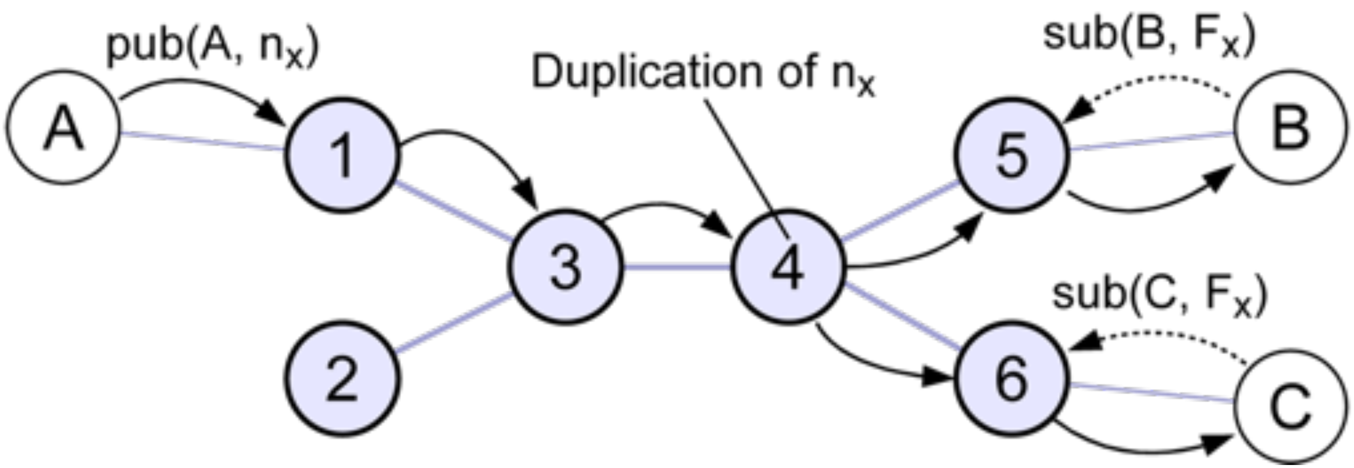
- The network of brokers forms an overlay network

- Routing can be split up into two layers
  - At the lower level, requests, i.e. control and data **messages** must be routed between brokers
  - At the higher level, **notifications** must be routed according to subscriptions and advertisements

- Routing algorithm depends on overlay structure
  - Unstructured, generic peer-to-peer networks must avoid routing messages in cycles, e.g., use
    - Variants of Distance Vector Routing
    - Spanning Tree
  - Structured peer-to-peer networks, e.g., use
    - Distributed Hash Tables

Routing: Principles

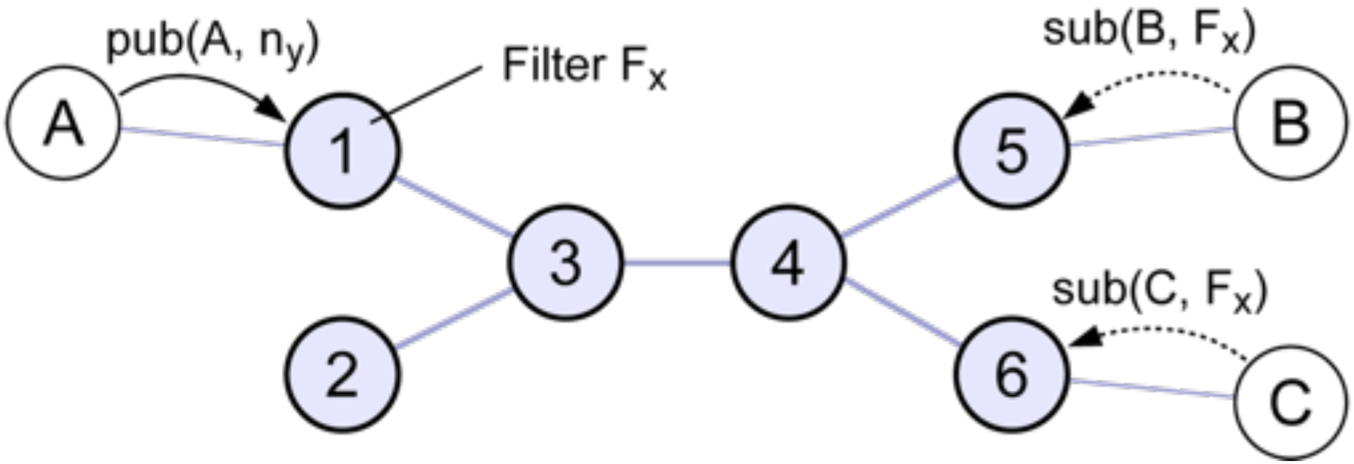
- Downstream duplication
  - Route notification as a single copy as far as possible



  - Clients B, C subscribe at routers 5, 6 with filter F_X
  - Client A publishes notification n_X (which is covered by F_X) to router 1
  - The notification is replicated not before router 4

Routing: Principles

- Upstream filtering
  - Apply filters upstream (as close as possible to source)



  - Clients B, C subscribe at routers 5, 6 with filter F_X
  - Client A publishes notification n_y (not covered by F_X) to router 1
  - The notification is discarded at router 1

Routing with Subscriptions

- Each broker maintains a routing table T_S to route notifications based on subscriptions

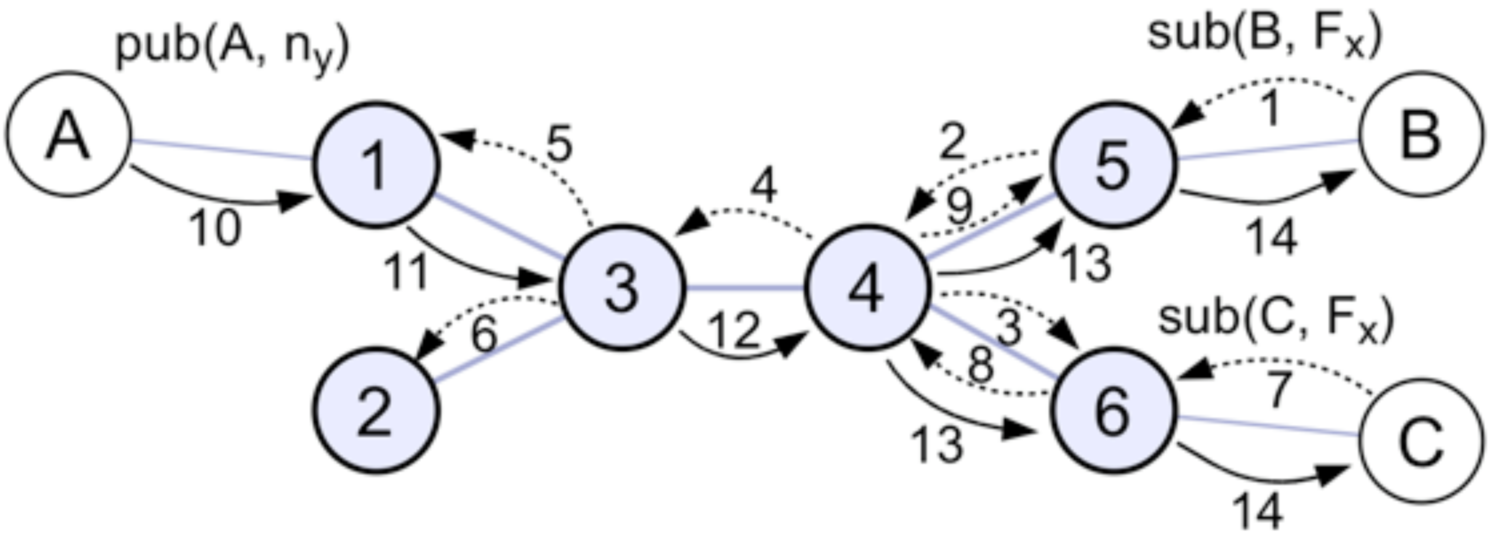
- **Routing of Notifications:** A notification n is only forwarded to a destination D, iff $\exists\ (D, F) \in T_S : n \sqsubseteq F$.

- **Routing of Subscriptions:** If a subscribe or unsubscribe request is received, the table T_S is updated accordingly.
  Subscribe or unsubscribe requests are potentially forwarded to all neighbors $D \neq E$, according to the underlying routing algorithm.
  - where *D* is the destination and *E* the source of the request

- Basic algorithm: Subscription is stored & forwarded from originating server to all servers in the network
  -> Tree that connects subscriber with each server

Routing with Subscriptions

- Example (with merging)



- **Filter merging** is used to reduce subscribe requests
- Routing paths for notifications are set by subscriptions
- Notifications routed towards subscriber following reverse path

Routing with Advertisements

- Basic Idea
  - Subscriptions are only forwarded towards publishers that intend to generate notifications that are potentially relevant to this subscription
  - Every advertisement is forwarded throughout the network, thereby forming a tree that reaches every server
  - Subscriptions are propagated in reverse, along the path to the advertiser, thereby **activating** the path
  - Notifications are then forwarded only through **activated** paths.

Routing with Advertisements

- Each broker maintains
  - a routing table T_S to route notifications based on subscriptions
  - a routing table T_A to route subscriptions based on advertisements

- **Routing of Notifications:** A notification n is only forwarded to a destination D, iff $\exists\ (D, F) \in T_S : n \sqsubseteq F$.

- **Routing of Subscriptions:** If a subscribe or unsubscribe request is received, the table T_S is updated accordingly. Subscribe or unsubscribe requests with a filter F_S are only forwarded to a broker D, iff $\exists\ (D, F_A) \in T_A : F_S \sqcap F_A$.

Routing with Advertisements

- **Routing of Advertisements:** If a broker receives a new advertisement with a filter F_A from a neighbor E, it
  - forwards all subscriptions to E that came from a destination $D \neq E$, overlap with F_A, and do not overlap with any previous advertisement from E:
    $\{(D, F_S) \in T_S \mid D \neq E \wedge F_S \sqcap F_A \wedge \neg\exists\ (D', F'_A) \in T_A : D' = E\ \wedge\ F_S \sqcap F'_A\}$
  - adds the advertisement to T_A:
    $T_A = T_A \cap \{(E, F_A)\}$
  - forwards the advertise request potentially to all neighbors $D \neq E$, according to the underlying routing algorithm.

- If a broker receives an unadvertisement request with a filter F_A from a neighbor E, it
  - removes the advertisement from T_A:
    $T_A = \{(D, F'_A) \in T_A \mid \neg(D = E\ \wedge\ F_A = F'_A)\}$
  - removes all routing entries from T_S of all neighbors $U \neq E$, for whose filter there is no other advertisement from any other destination $D \neq U$ that overlaps:
    $T_S = T_S - \{(U, F_S) \in T_S \mid U \neq E \wedge \neg\exists\ (D, F'_A) \in T_A : D \neq U \wedge F_S \sqcap F'_A\}$
  - forwards the unadvertise request potentially to all neighbors $D \neq E$, according to the underlying routing algorithm.

Scalability

- System should be scalable in terms of
  - the number of clients (i.e., producers and consumers),
  - the number of event routers,
  - the number of subscriptions and advertisements, and
  - the amount of traffic (e.g., number of notifications/second)

- Problems in unstructured peer-to-peer overlays
  - Either subscriptions or advertisements forwarded to each node
    - Assumption (for Internet-based services): Advertisements are rather static, subscriptions are dynamic

      -> Use routing with advertisements
  - Routing tables grow proportionally with the size of the network

    -> use filter merging

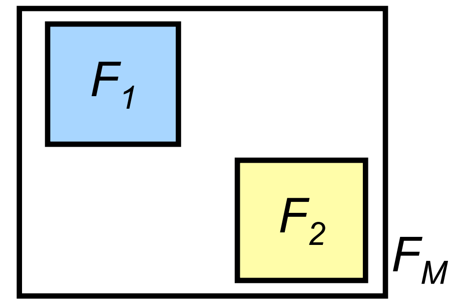
    -> use structured overlays

Filter Merging

- Inexact Merging

F_M is an inexact merge of F_1 and F_2 iff

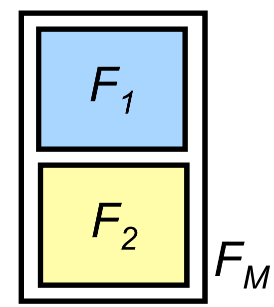
$$F_M \sqsupseteq F_1 \wedge F_M \sqsupseteq F_2$$



- Exact Merging

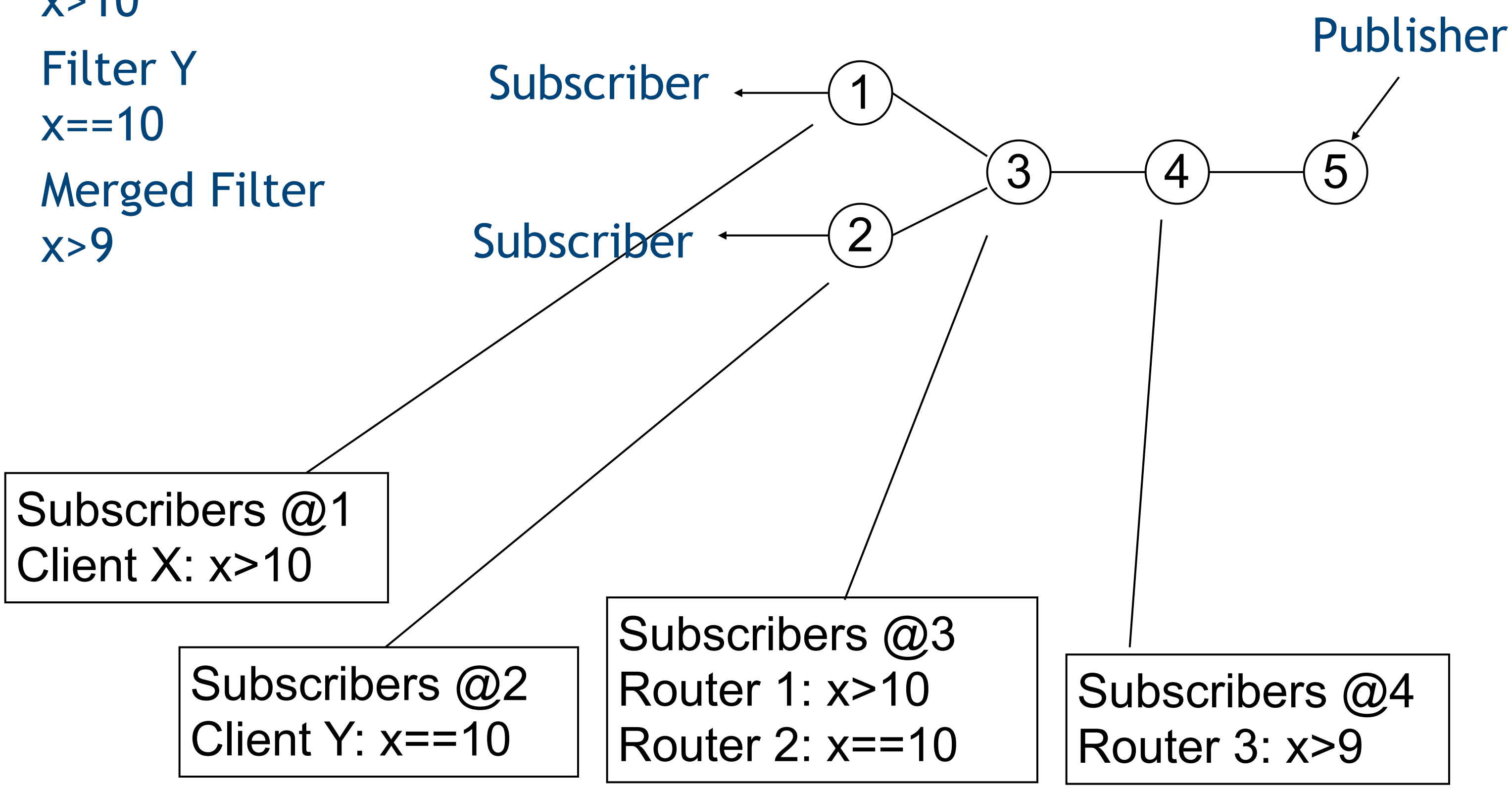
F_M is an exact merge of F_1 and F_2 iff

$$F_M \sqsupseteq F_1 \wedge F_M \sqsupseteq F_2 \wedge \neg\exists F_3 : (F_3 \not\sqcap F_1 \wedge F_3 \not\sqcap F_2 \wedge F_M \sqsupseteq F_3)$$



Filter Merging: Example

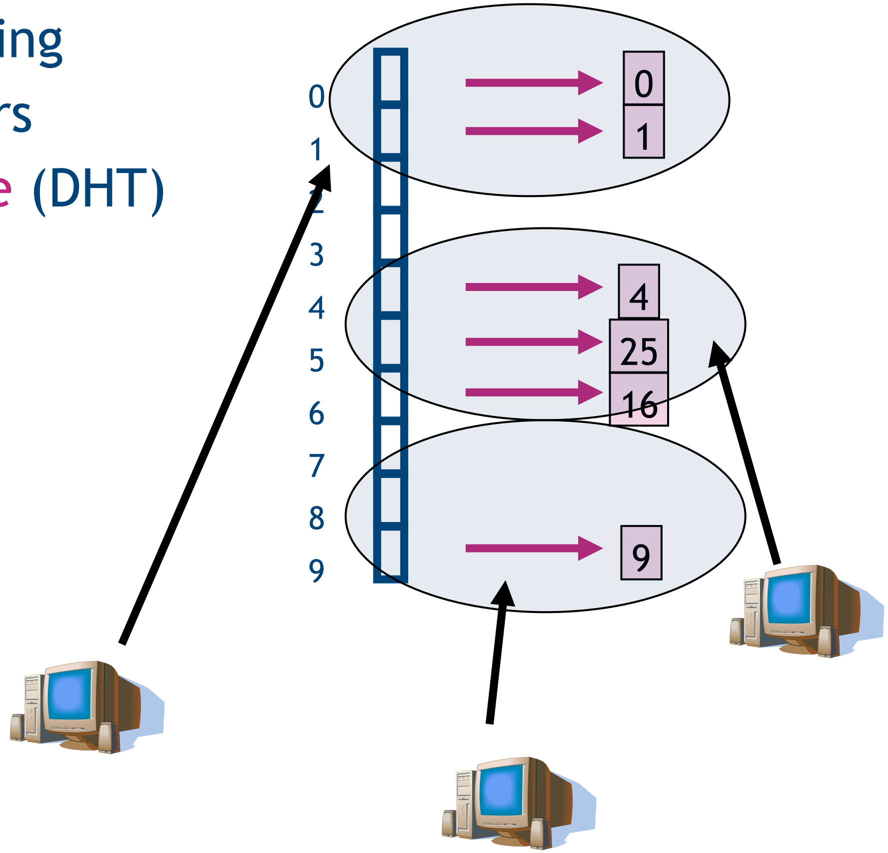
- Filter Merging
  - Filter X
    x>10
  - Filter Y
    x==10
  - Merged Filter
    x>9



Structured Overlays

- Basic Idea
  - Hash tables are fast for searching
  - Distribute hash buckets to peers
  - Result is Distributed Hash Table (DHT)

- Example:
  - Hash function:
    hash(x) = x mod 10
  - Insert numbers 0, 1, 4, 9, 16, and 25



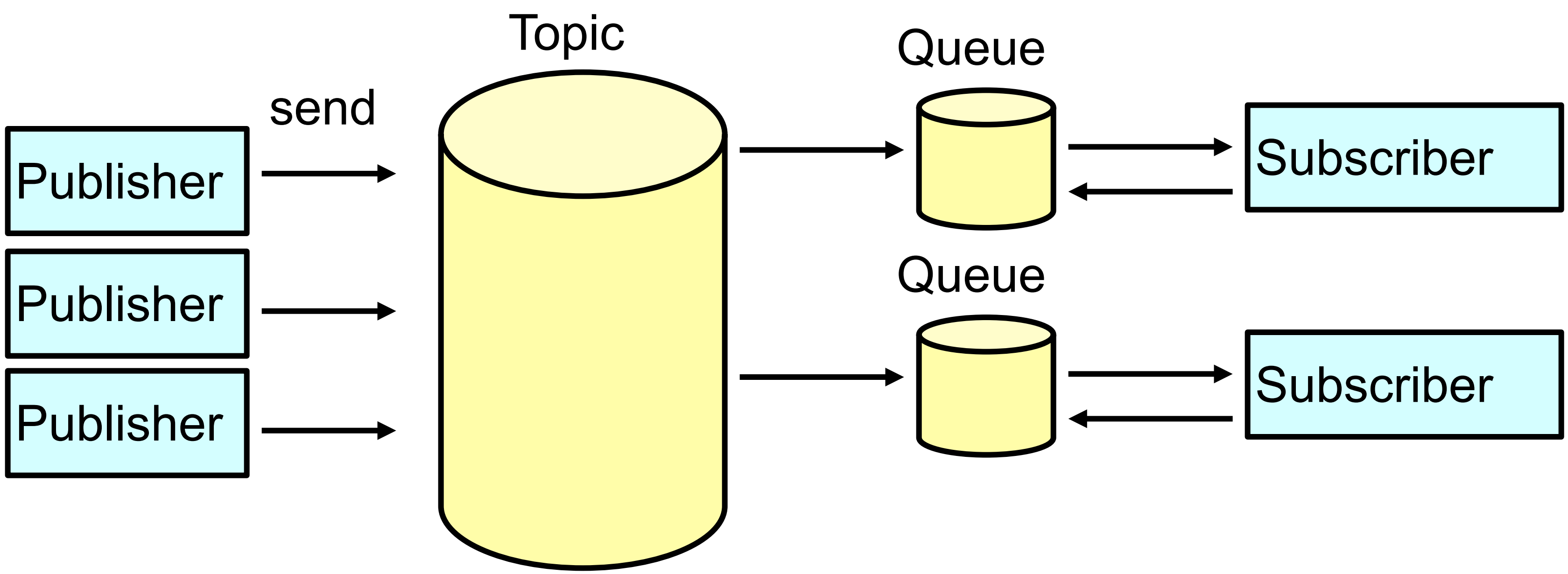
Structured Overlays

- Systems based on Distributed Hash Tables (e.g. SCRIBE)
- In a DHT, the storage location of an information item is defined by its hash value
  - Channel-based addressing: calculate hash value from channel name
  - Content-based addressing: no general solution
    - -> „**Channelization**“: calculate hash from selected attributes, e.g. message type
- The (global) subscription table is distributed over the network
  - A broker is responsible for specific subscriptions
  - The broker is the rendezvous point for publishers and subscribers
- Routing of subscriptions
  - Subscriber calculates hash of subscription h(S) and sends it to the broker with hash h(B) closest to h(S). The subscription is stored at B.
- Routing of notifications
  - Publisher calculates hash of notification h(n) and sends it to the broker with h(B) closest to h(n). Broker B has a list of all relevant subscribers.

QoS and Transactions

- Quality of Service
  - Guaranteed delivery
    - Logistics
    - Stock quotes
  - Low latency
    - sensor, audio, or video data streams
- Local Transactions
  - between the publisher and the event service, or between the event service and the subscriber
  - groups a series of operations into an atomic unit of work

Mobility Support: Durable Subscriptions



- Messages are stored for each subscriber
- Permits disconnection of subscriber
  - But: Subscriber bombarded with messages on reconnect (Remedy: Use TTL)

System Examples

- Industry-strength
  - JMS
  - CORBA Notification Service
  - Elvin
  - IBM WebSphere MQ Event Broker (Gyphon)

- Research Prototypes
  - REBECA
  - SIENA
  - Gryphon

JMS: Java Message Service

- API
  - „Common set of interfaces and associated semantics“
- Domains
  - Point-to-Point: Message-Queue
  - Publish/Subscribe
    - Topic-based
    - Subject-based
    - Durable Subscribers
- Separated Administration
  - Queues and Topics are created with product-specific administration tools
  - Application independent
  - Local Transactions

JMS: Java Message Service

- Message Format
  - Header: Predefined Fields (ID, Destination, Timestamp, Priority)
  - Properties (optional): Accessible for Filtering
    Values can be boolean, byte, int, ... double and String
  - Body (optional): Five Types
    - TextMessage: String (XML Document)
    - MapMessage: Key/Value-Pairs
    - BytesMessage: Stream of uninterpreted bytes
    - StreamMessage: Stream of primitive values
    - ObjectMessage: A serializeable object
- Event Consumption
  - Synchronously: Subscriber explicitly fetches message from destination
  - Asynchronously: Subscriber registers a message listener

JMS: Message Filtering

- SQL92 conditional expressions (Limited)
  - Logical operators in precedence order: NOT, AND, OR
  - Comparison operators: =, >, >=, <, <=, <> (not equal)
  - Arithmetic operators in precedence order: +, - (unary) *, / (multiplication and division) +, - (addition and subtraction)
  - arithmetic-expr1 [NOT] BETWEEN arithmetic-expr2 AND arithmetic-expr3 (comparison operator)
  - identifier [NOT] IN (string-literal1, string-literal2,...) (comparison operator where identifier
  - identifier [NOT] LIKE pattern-value [ESCAPE escape-character]
  - identifier IS [NOT] NULL (comparison operator that tests for a null header field value or a missing property value)
- Examples:
  - NewsType='Opinion' OR NewsType='Sports'
  - phone LIKE '12%3'
  - JMSType='car' AND color='blue' AND weight>2500

JMS: Implementations

J2EE Licensees:

- Allaire Corporation: JRun Server 3.0
- BEA Systems, Inc.: WebLogic Server 6.1
- Brokat Technologies (formely GemStone)
- IBM: MQSeries
- iPlanet (formerly Sun Microsystems, Inc. Java Message Queue)
- Oracle Corporation
- SilverStream Software, Inc.
- Sonic Software
- SpiritSoft, Inc. (formerly Push Technologies Ltd.)
- Talarian Corp.

Open source:

- Apache ActiveMQ
- objectCube, Inc.
- OpenJMS
- ObjectWeb – Joram
- …

Selected other companies:

- SwiftMQ
- Fiorano Software
- Nirvana (PCB Systems)
- Orion
- SeeBeyond
- Software AG, Inc.
- SoftWired Inc.
- Sunopsis
- Venue Software Corp.

Under development:

- Novosoft, Inc. (vendor implementation)
- spyderMQ (open-source, email interest group)
- …

JMS: SwiftMQ

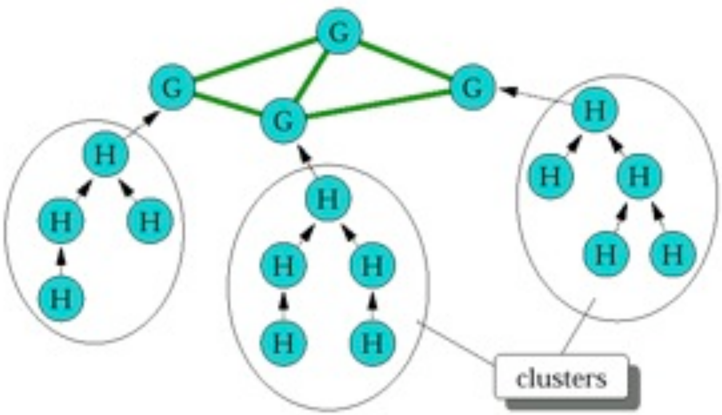
- Domain
  - Point-to-Point
  - Topic- and Subject-based Publish/Subscribe
- Server Topology
  - Generic Peer-to-Peer: Federated Router Network
- Features
  - Fully implements JMS 1.0.2 Specification
  - Topic Hierarchies
  - SQL-Like Predicate Topic Addressing
    Permits subscription with topic name wildcard. Example:

    iit.s%s._S matches iit.sales.US
  - File based persistent message store

Dr. Erwin Aitenbichler
Prof. Dr. M. Mühlhäuser
Telekooperation
©

SIENA

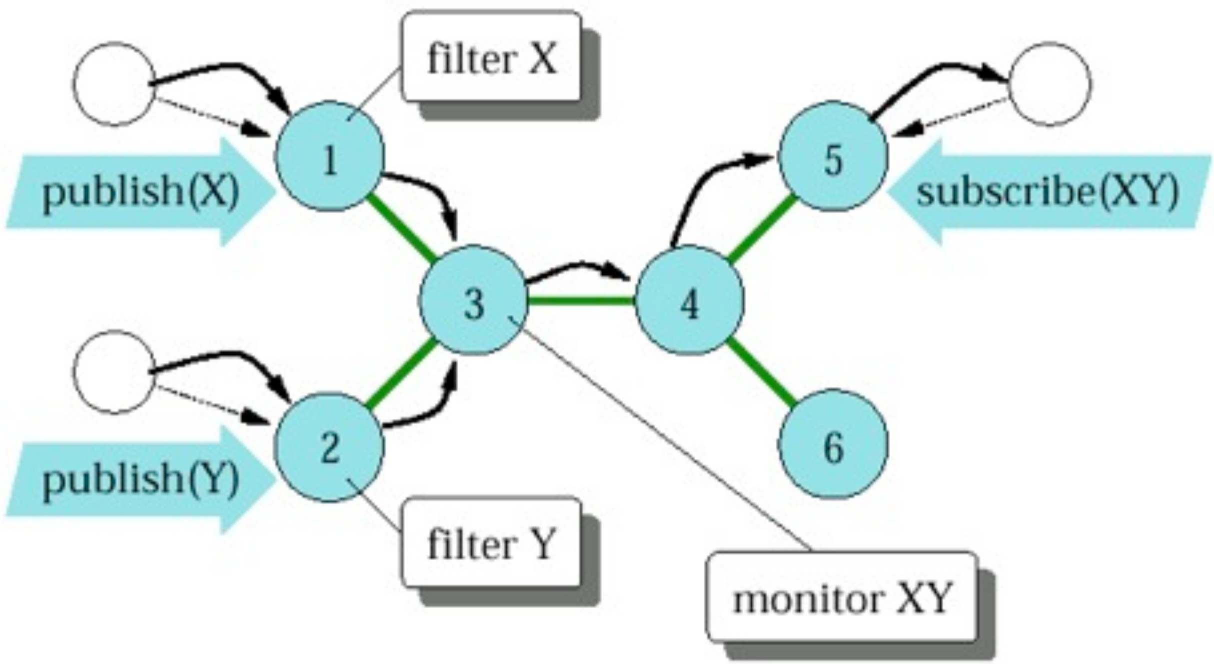
- SIENA = Scalable Internet Event Notification Architecture
- Domain
  - Advertisement-based Publish/Subscribe
  - Content-based Subscriptions
- Server Topology
  - Generic Peer-to-Peer
  - Hybrid topology
    - LAN: Hierarchical
    - WAN: Generic Peer-to-Peer



- Data Model
  - Notification is set of attribute=(name, type, value)
  - Limited set of types (string, time, date, integer, float, ...)
- Subscription Language
  - Filter is set of attr_filter=(name, type, operator, value)
  - Operators: any, =, <, >, >* (prefix), *< (postfix)
  - Pattern Monitoring (Temporal sequence of events)

SIENA: Routing Strategies

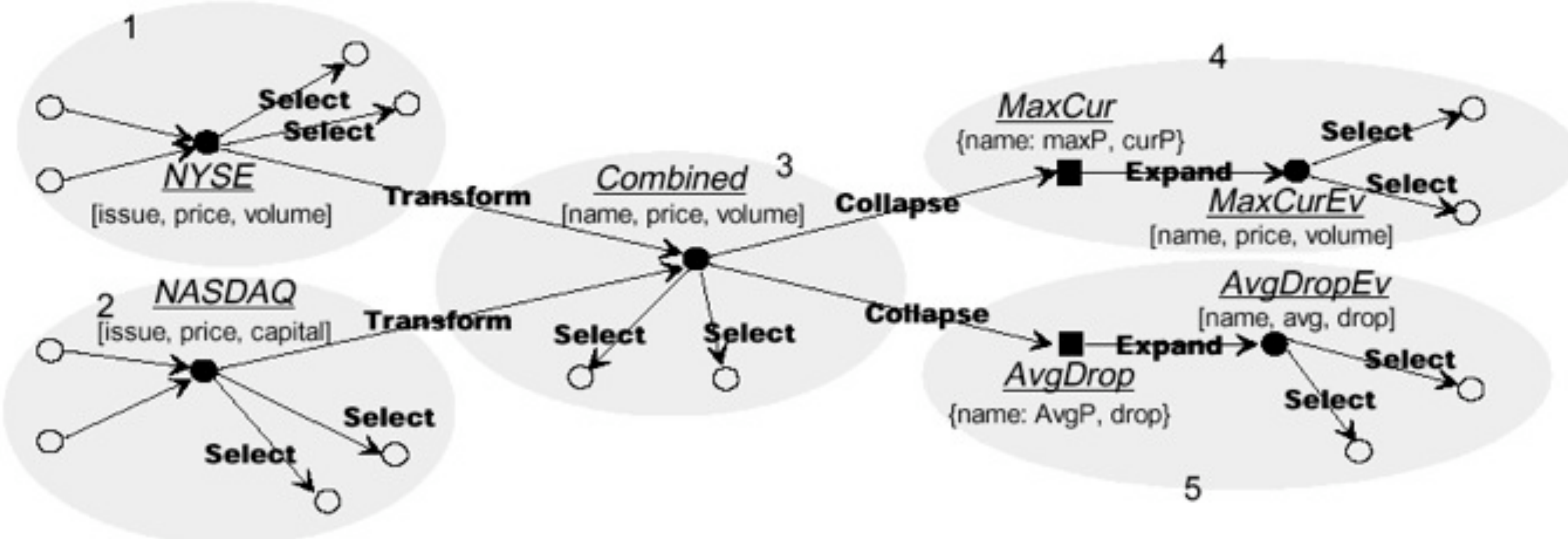
- Monitoring
  - Detects temporal sequences of events
  - Apply filters upstream (as close as possible to source)
  - Assemble patterns upstream



Elvin

- Elvin3
  - Subscription-based Publish/Subscribe
  - Content-based Subscriptions
  - Centralized Server
- Elvin4
  - Data Model: Typed Key/Value-Pairs
    Types: integer (32 and 64), string, FP, binary data (opaque)
  - Subscription Language:
    Simple Integer and FP arithmetic
    Strings: POSIX ERE (Extended Regular Expressions),
    begins-with, ends-with, contains (for better optimization)
  - Quenching
    - Mechanism for publisher to determine whether subscribers are interested in their messages
    - Auto-Quenching (Appears to be subset of SIENA Sub.Fwd.)
  - Source code available for non-commercial use
  - Proxy at network boundary to support disconnection

Gryphon: Information Flow Graphs



- Information providers and consumers
- Information Spaces: Event histories (NYSE) or states (MaxCur)
- Dataflows: Directed Edges, four types:
  - **Select**: Connects two histories with same schema, filter predicate
  - **Transform**: Transforms from one schema to another
  - **Collapse**: Connects history to state, collapse rule
  - **Expand**: Inverse of collapse

Pub/Sub: Summary

- Loosely coupled systems
  - Space decoupling
  - Time decoupling
  - Control flow decoupling
- Publish/Subscribe
  - Powerful and scalable abstraction for decoupled interaction
  - Problems are at the algorithm & implementation level
  - Research Challenges: Scalability/Expressiveness-Tradeoff, Fault Tolerance, Integration w. P2P, Security, Reliability, …
- Has specific application areas
  - e.g., RFID middleware, sensor systems in Ubicomp
  - will not replace request/reply, etc. (in all areas)

Pub/Sub: Literature

- Gero Mühl, Ludger Fiege, Peter R. Pietzuch:
  Distributed Event-Based Systems
  Springer Verlag, ISBN: 978-3540326519

- Erwin Aitenbichler:
  Event-Based and Publish/Subscribe Communication
  Chapter VII in Mühlhäuser, Gurevych: Ubiquitous Computing Technology for Real Time Enterprises
  ISR, ISBN: 978-159904832-1