Cloud computing and distributed systems

Chapter #6
Indirect communication

Zeynep Yücel

Ca' Foscari University of Venice
zeynep.yucel@unive.it
yucelzeynep.github.io

Indirect communication

- Communication through an intermediary: No direct coupling of sender and receiver
- Flexibility, easier management.
- Used in environments where change is anticipated (e.g. mobile environments).
- Useful for event dissemination.
- Main disadvantage: Performance overhead.

Indirect communication

	Time-coupled	*Time-uncoupled*
Space coupling	*Properties*: Communication directed towards a given receiver or receivers; receiver(s) must exist at that moment in time *Examples*: Message passing, remote invocation	*Properties*: Communication directed towards a given receiver or receivers; sender(s) and receiver(s) can have independent lifetimes
Space uncoupling	*Properties*: Sender does not need to know the identity of the receiver(s); receiver(s) must exist at that moment in time *Examples*: IP multicast	*Properties*: Sender does not need to know the identity of the receiver(s); sender(s) and receiver(s) can have independent lifetimes *Examples*: Most indirect communication paradigms covered in this chapter

- Key properties:
  - Space uncoupling: identities irrelevant.
  - Time uncoupling: independence from lifetime.

Indirect communication

A closer look at space and time uncoupling

- Not necessarily uncoupled both in space and time.
  - IP multicast: space-uncoupled, time-coupled.
- Indirect communication: all paradigms involving an intermediary.
  - Coupling levels vary.
- Distinction between asynchronous communication and time uncoupling:
  - Asynchronous: sender is not blocked.
  - Time uncoupling: independent existence of sender and receiver.

Group communication

- Group communication is an example of indirect communication.
- Abstraction over multicast communication.
- Send messages without knowledge of individual identities.
- Key applications:
  - Reliable dissemination to a large number of clients.
  - Support for collaborative application.
  - Fault tolerance.

Group communication

The programming model

- Multicast to all members.
  - Dynamic group membership.
  - Messages propagated with certain guarantees reliability and ordering.
    - Broadcast to all members.
    - Unicast to one member.
- Advantages:
  - Better bandwidth utilization.
  - Send once over any link.
  - Reduced transmission time.
  - Consistence in delivery guarantees.

Group communication

Process groups and object groups

- Majority takes place in process groups.
- Process groups with low-level service:
  - Basic message delivery (without advanced dispatching).
  - Unstructured messages.
- Object groups:
  - Higher-level approach.
  - Collection of objects.
  - Concurrent invocations.
  - Client-proxy interaction.
    - Clients unaware of replication.
  - Proxy communicates with group.
  - Parameters, results processed similar to RMI.

Group communication

Other key distinctions

- Open vs. closed groups:
  - Closed: limited to members.
  - Open: unlimited.
- Overlapping vs. non-overlapping groups:
  - Overlapping: Possible to be a member of multiple groups.
  - Non-overlapping: single membership.
- Synchronous or asynchronous groups.
- These characteristics influence the design of multicast algorithm.

Group communication

Implementation issues and reliability and ordering in multicast

- Reliability of one-to-one communication requires:
  - Integrity: the messages are delivered correctly and at most once.
  - Validity: a message sent will eventually be delivered
- Reliability of GC has a third property:
  - Agreement: if one member of a group receives a message, all must receive it.

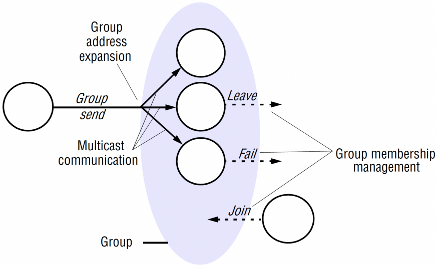
Group communication

Implementation issues and reliability and ordering in multicast

- Ordering: GC guarantees specific orders:
  - FIFO Ordering: Message delivery order.
    - Process A sends m1, m2; B receives m1, m2.
    - No guarantees on the order of messages from A to B and relative to messages from C to B.
  - Causal Ordering: Causality respected.
    - Independent messages can be delivered arbitrarily.
  - Total Ordering: Consistent global order.
    - All messages seen identically by all members.

Group communication

Group Membership Management



- Group membership service: Maintains member view.
- Main tasks:
  - Interface for changes: Manage group and process adjustments.
  - Failure detection: Monitor and mark faulty members.
  - Notification: Inform about membership changes.
  - Address Expansion: Group IDs to member lists.

Group communication

Group Membership Management

- IP Multicast: Basic membership service.
  - Dynamic group join/leave.
  - Single IP for group.
  - No membership info provided.
  - Delivery not aligned with changes.
  - Full properties need *view-synchronous communication*.
- Challenges: Membership maintenance issues.
  - Impacts group-based effectiveness.
  - Best for small, static systems.
  - Less effective in volatile environments.

Publish-subscribe systems

- Popular form of indirect communication.
  - Distributed event based systems.
  - Event communication applications.
- Publishers share structured events.
- Subscribers express interest via subscriptions.
- System matches subscriptions to events.
- One-to-many model communication.

Publish-subscribe systems

Applications of publish-subscribe systems

- Widely used in:
  - Financial systems.
  - Real-time data feeds.
  - Cooperative environments.
  - Ubiquitous computing.
  - Monitoring applications.
- Example: dealing room system.

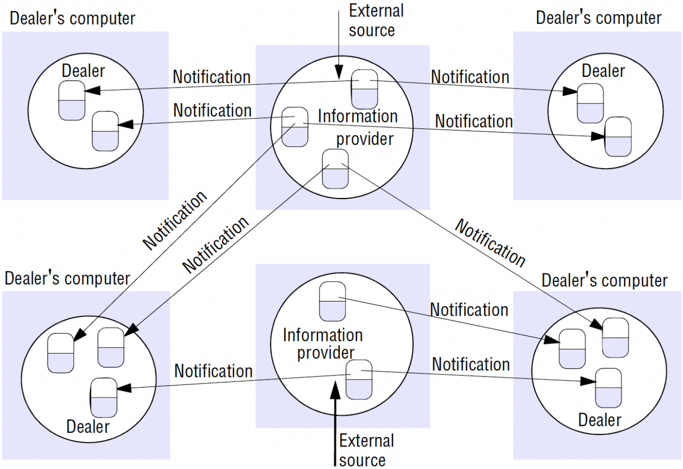
Publish-subscribe systems

Dealing room system

- Dealing room: Latest market prices.
- Stocks (market prices) as objects.
- Information from external sources.
- Dealers focus on specific stocks.
- Main processes:
  - Information Provider: Receives and publishes updates.
  - Dealer Process: Creates subscriptions and displays info.

Publish-subscribe systems

Dealing room system illustration



Publish-subscribe systems

Characteristics of publish-subscribe systems

- Heterogeneity:
  - No need to design for interoperability.
  - Components work together via event notifications.
- Asynchronicity:
  - Notifications sent independently.
  - Independent operation.

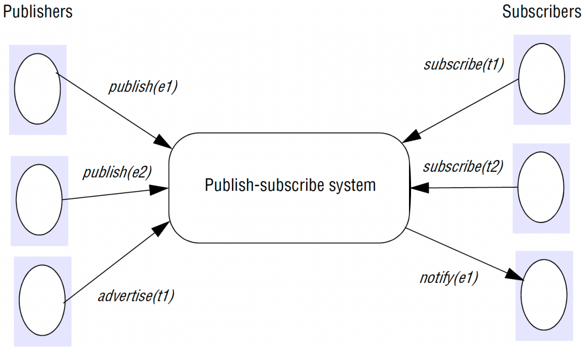
Publish-subscribe systems

Characteristics of publish-subscribe systems

- Delivery Guarantees:
  - Different applications require varying notification reliability.
- Real-Time Requirements:
  - Certain applications need real-time guarantees.

Publish-subscribe systems

The programming model



- Publish-Subscribe Model operations:
  - Publisher disseminates event e: ‘publish(e)‘.
  - Subscribers express interest with filter f: ‘subscribe(f)‘.
  - Subscribers revoke interest: ‘unsubscribe(f)‘.
  - Events are delivered to subscribers: ‘notify(e)‘.
  - Publishers declare nature of future events: ‘advertise(f)‘.
  - Publishers revoke advertisements: ‘unadvertise(f)‘.

Publish-subscribe systems

The programming model

- Subscription Models:
  - Channel-based: Named channels.
    - Simplicity: Straightforward.
    - Limitation: No content filtering.
  - Topic-based: Categorized by topics.
    - Hierarchy: Hierarchical topic organization.
    - Difference: Explicitly defined topics.
    - Flexibility: Broad or specific interests.

Publish-subscribe systems

The programming model

- Content-based Approaches: Multiple attribute filtering.
  - Query Expression: Attribute constraints used.
  - Expressiveness: More expressive filtering.
  - Complexity: Sophistication due to query.
- Type-based Approaches: Defined by event types.
  - Filtering Range: Broad or fine filtering.
  - Expressiveness: Fine-grained capabilities.
  - Focus: Structure and types emphasized.

Publish-subscribe systems

The programming model

- Key points and differences:
  - Channel-based: Simple, no filtering.
  - Topic-based: Explicit, hierarchical topics.
  - Content-based: Advanced, expressive filtering.
  - Type-based: Filters by event structures.

Publish-subscribe systems

The programming model

Other subscription categories

- Commercial systems focus on objects of interest.
  - Focus on state changes.
  - Objects react to changes.
    - E.g. interactive applications handle user events.
  - Objects receive notifications about changes.

Publish-subscribe systems

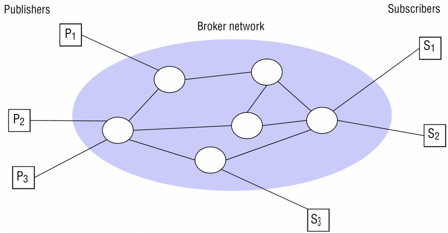
Implementation issues and Centralized versus distributed implementations

- Principal role: Deliver events based on filters.
- Other issues to handle: security, scalability, etc..
  - Complex to achieve all goals at once.
- Implementation types:
  - Centralized
  - Distributed
  - Peer-to-peer

Publish-subscribe systems

Implementation issues and Centralized versus distributed implementations

- Centralized: Single node broker.
  - Publishers send events to broker.
  - Simple, but lacks resilience.

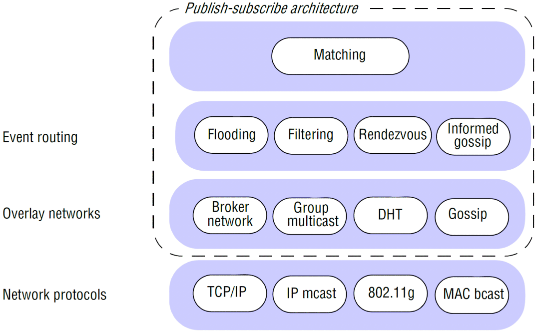


- Distributed: Network of brokers.
  - Survives failures.
  - Scalable for large use.

- Peer-to-peer implementations: No distinction between nodes.
  - All nodes act as brokers.

Publish-subscribe systems

Overall systems architecture



- Bottom layer: Various communication services (e.g. TCP/IP).
- Event routing directs notifications.
  - Overlay infrastructure supports routing.
- Top layer implements matching for subscriptions.

Publish-subscribe systems

Implementation approaches - Flooding

- Flooding is simple.
  - Subscribers handle event matching.
  - Subscriptions sent to publishers.
    - Matched events sent directly.
- Flooding uses broadcast or multicast.
- Brokers create acyclic graph for notifications.
- Simple, but excessive traffic occurs.

Publish-subscribe systems

Implementation approaches - Filtering

- Filtering-based routing forwards notifications.
  - Subscription information propagated.
  - Brokers maintain three types of information:
    - Neighbours list,
    - Subscription list,
    - Routing table.
  - Brokers use a matching function.
    - Notifications and subscriptions.
  - Brokers also handle incoming subscriptions.
    - Updates subscriptions or routing table.

Publish-subscribe systems

Overall systems architecture - Advertisements and Rendezvous

- Excessive traffic
- Load balancing
  - Broker assignment
  - Broker identification
  - Non-empty intersection
- Gossip methods

Publish-subscribe systems

Examples of publish-subscribe systems

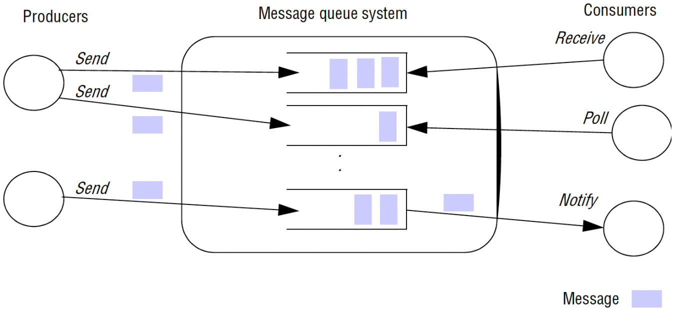
System (and further reading)	*Subscription model*	*Distribution model*	*Event routing*
CORBA Event Service (Chapter 8)	Channel-based	Centralized	-
TIB Rendezvouz [Oki *et al.* 1993]	Topic-based	Distributed	Ffiltering
Scribe [Castro *et al.* 2002b]	Topic-based	Peer-to-peer (DHT)	Rendezvous
TERA [Baldoni *et al.* 2007]	Topic-based	Peer-to-peer	Informed gossip
Siena [Carzaniga *et al.* 2001]	Content-based	Distributed	Filtering
Gryphon [www.research.ibm.com]	Content-based	Distributed	Filtering
Hermes [Pietzuch and Bacon 2002]	Topic- and content-based	Distributed	Rendezvous and filtering
MEDYM [Cao and Singh 2005]	Content-based	Distributed	Flooding
Meghdoot [Gupta *et al.* 2004]	Content-based	Peer-to-peer	Rendezvous
Structure-less CBR [Baldoni *et al.* 2005]	Content-based	Peer-to-peer	Informed gossip

Message queues

- Point-to-point communication
  - Message placed in a queue and then retrieved
  - Space-time uncoupling

Message queues

The programming model



- Producers send messages, consumers receive them.
- Receive styles
  - Blocking
  - Non-blocking
  - Notify

Message queues

The programming model

- Many processes can send to or receive from a queue.
- Typically FIFO order, but priority or selection are also possible
- Message attributes
  - Destination
  - Metadata
  - Body
- Configurable sizes
  - Serializable in various formats

Message queues

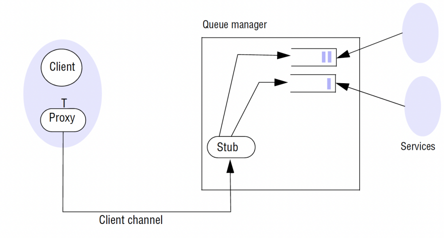
The programming model

- Persistence
  - Indefinite storage until consumption
  - Disk commitment
- Reliable communication
  - Validity
  - Integrity
- Other properties
  - Transaction support
  - Message transformation
  - Security features

Message queues

Implementation issues and Case study: WebSphere MQ

- Centralized or distributed
  - Centralized is simple with a risk of bottleneck
- WebSphere MQ by IBM



  - Queue managers
    - Host and manage queues
  - Remote access
  - Proxies used

Message queues

Implementation issues and Case study: WebSphere MQ

- Interconnected managers
  - Message channels
  - Client channels
- Customized topologies

Message queues

Hub-and-spoke approach

- Main queue manager
- Indirect client connection
  - Message forwarding
- Strategic placement
- Reduce latency
- RPC communication
- Bottleneck risk

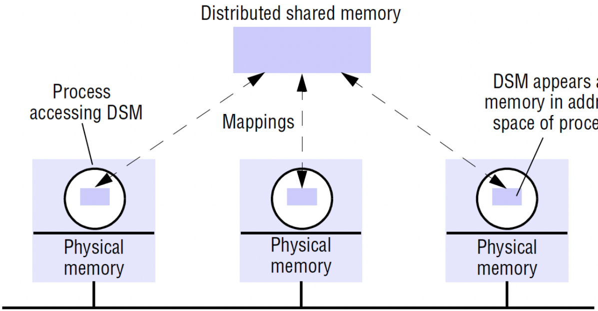
Shared memory approaches

Parallel and distributed

- Parallel computing
  - Multiple processors collaborate.
  - Shared resources accelerate tasks.
- Distributed Computing
  - Independent machines communicate.
  - Local resources coordinate tasks.
- What is Distributed shared memory?
  - Data sharing across computers.
- Distributed shared memory techniques and tuple space communication.

Shared memory approaches

Distributed Shared Memory (DSM)



- DSM is an abstraction for data sharing.
- Processes access DSM like regular memory.
  - System ensures process updates.
  - Accessing as a single shared memory.

Shared memory approaches

Distributed Shared Memory (DSM)

- DSM reduces message passing.
- Message passing still required.
    - Manage replicated data locally.

Shared memory approaches

Comparing DSM to message passing (MP)

- Data Communication Method
  - DSM: Direct variables sharing.
  - MP: Requires marshalling.
- Process Isolation
  - DSM: Shared memory risks.
  - MP: Private address spaces.

Shared memory approaches

Comparing DSM to message passing (MP)

- Handling Heterogeneity
  - DSM: Difficulties due to differences in data representation.
  - MP: Marshalling accommodates differences.
- Process Lifetimes
  - DSM: Non-overlapping lifetimes are fine.
  - MP: Requires simultaneous activity.

Shared memory approaches

Comparing DSM to message passing (MP)

- Performance:
  - Similar performance in parallel programs.
  - Generalization is difficult.
- Cost Visibility:
  - In MP, costs are explicit.
  - DSM costs may be unclear.
- Application Suitability:
  - No clear answer for all applications.
  - Choice depends on requirements.

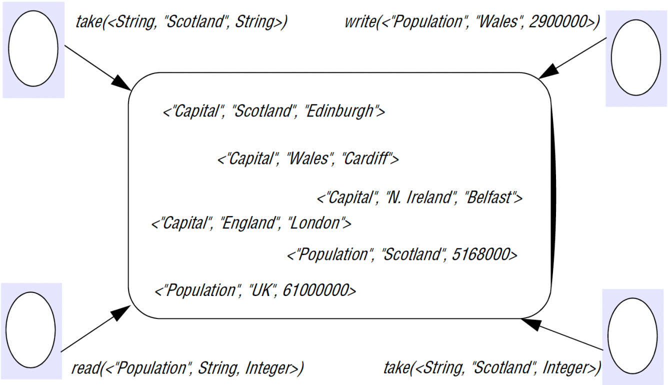
Shared memory approaches

Tuple space communication

- Tuples are typed data sequences.
- Processes communicate via tuple space.
- Tuples accessed by pattern matching.
- Tuples are immutable.
  - Must be removed and replaced to change.

Shared memory approaches

Tuple space communication



- Processes can write, read, take tuples.
  - Write: Adds without removing.
  - Read: Retrieves without affecting.
  - Take: Retrieves and removes.

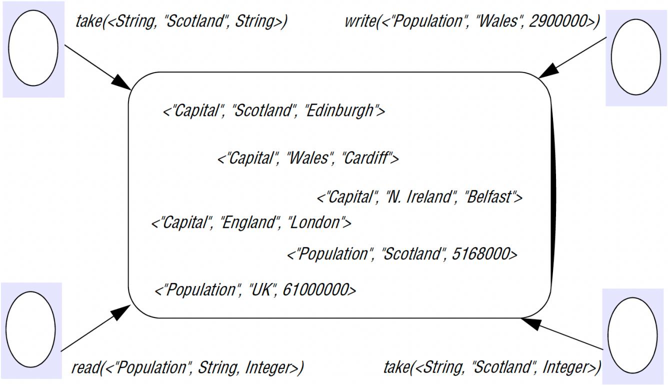
Shared memory approaches

Tuple space communication

- Tuple space reads/removes tuples specifications.
  - Returns tuples via associative addressing.
  - Tuples lack addresses.
- Blocking read/take operations until match.
- Tuple specification defines fields.
  - Example: $take(\langle String, integer \rangle)$ extracts tuples.

Shared memory approaches

Tuple space communication



- $take(\langle String, "Scotland", String\rangle)$
- $take(\langle String, "Scotland", Integer\rangle)$
- $write(\langle "Population", "Wales, 2900000\rangle)$ inserts tuple.
- $read(\langle "Population", String, Integer)$
    - Non-deterministic selection by implementation.

Shared memory approaches

Tuple space communication

- Tuple space properties:
  - Space Uncoupling: Various senders/recipients.
  - Time Uncoupling: Operate at different times.

Shared memory approaches

Implementation issues

- Centralized server simple, not scalable.
- Proposed distributed solutions.
- Replication improves reliability.
  - 1. State machine approach:
    - Tuple space as state machine.
    - Consistency via deterministic reactions.

Shared memory approaches

Implementation issues - Xu and Liskov's approach

- 2. Xu and Liskov's approach optimizes replication.
  - Updates in current view.
  - Tuples partitioned by logical names.

Shared memory approaches

Implementation issues - Xu and Liskov's approach

- Write: Multicast
  - Repeated until all acknowledgements received.
  - Duplicate requests need to be detected.
- Read: Multicast
  - Replicas search for matching tuples.
  - First match returned.

Shared memory approaches

Implementation issues

- Take operation complex, two-phase.
  - Phase-1: Specification sent, replicas lock.
  - Phase-2: Remove tuples from all replicas.

Shared memory approaches

Implementation issues

write
1. The requesting site multicasts the *write* request to all members of the view;
2. On receiving this request, members insert the tuple into their replica and acknowledge this action;
3. Step 1 is repeated until all acknowledgements are received.

read
1. The requesting site multicasts the *read* request to all members of the view;
2. On receiving this request, a member returns a matching tuple to the requestor;
3. The requestor returns the first matching tuple received as the result of the operation (ignoring others);
4. Step 1 is repeated until at least one response is received.

take

Phase 1: Selecting the tuple to be removed
1. The requesting site multicasts the *take* request to all members of the view;
2. On receiving this request, each replica acquires a lock on the associated tuple set and, if the lock cannot be acquired, the *take* request is rejected;
3. All accepting members reply with the set of all matching tuples;
4. Step 1 is repeated until all sites have accepted the request and responded with their set of tuples and the intersection is non-null;
5. A particular tuple is selected as the result of the operation (selected randomly from the intersection of all the replies);
6. If only a minority accept the request, this minority are asked to release their locks and phase 1 repeats.

Phase 2: Removing the selected tuple
1. The requesting site multicasts a *remove* request to all members of the view citing the tuple to be removed;
2. On receiving this request, members remove the tuple from their replica, send an acknowledgement and release the lock;
3. Step 1 is repeated until all acknowledgements are received.

Shared memory approaches

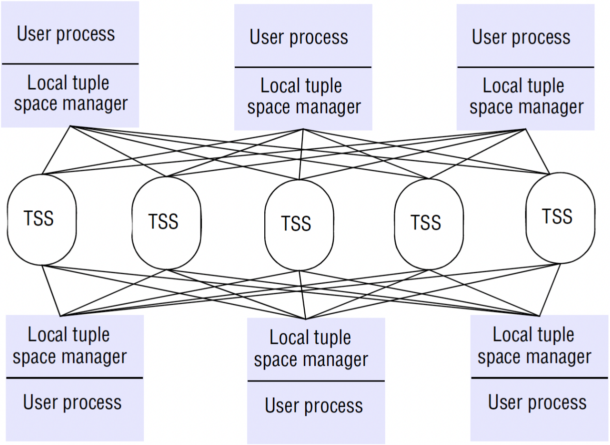
Implementation issues

- Key points:
  - Read blocks until first response.
  - Take blocks until phase 1.
  - Write operations return immediately.
- Concurrence issues require constraints.

Shared memory approaches

Implementation issues

- Various methods for tuple space abstraction.
- Linda Kernel partitions tuples across multiple TSSs.



Shared memory approaches

Implementation issues

- Hashing algorithm selects TSS for tuples.
- Reading/taking involves searching servers.
- Single tuple copy simplifies take.

Discussion topic

Message passing is both time- and space-coupled – that is, messages are both directed towards a particular entity and require the receiver to be present at the time of the message send. Consider the case, though, where messages are directed towards a name rather than an address and this name is resolved using DNS. Does such a system exhibit the same level of indirection?

Discussion topic

If a communication paradigm is asynchronous, is it also time-uncoupled? Explain your answer with examples as appropriate.

Discussion topic

In the context of a group communication service, provide example message exchanges that illustrate the difference between causal and total ordering