Indirect Communication

Today

- Space and time (un)coupling
- Group communication, pub/sub, message queues and shared memory

Next time

- Distributed file systems
Indirect communication

- Indirect communication – communication between entities through an intermediary with no direct coupling between senders/receivers.

- Two forms of uncoupling
  - Space – No need to know the identity of the other party, so you can change, update, replicate, move senders/receivers
  - Time – No need to exist at the same time, so it’s ok if you either gets disconnected for a bit
    - Not the same as asynchronous; with time uncoupling receiver doesn’t have to even exist when msg is sent
## Space and time coupling

<table>
<thead>
<tr>
<th>Space coupling</th>
<th>Time-coupled</th>
<th>Time-uncoupled</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Communication directed to a given receiver(s) that must be available at the time</td>
<td>Sender(s) and receiver(s) can have independent lifetimes</td>
</tr>
<tr>
<td></td>
<td>e.g. IP multicast</td>
<td>Sender does not need to know ID of receiver; sender(s) and receiver(s) can have independent lifetimes</td>
</tr>
<tr>
<td>Space uncoupling</td>
<td>Sender does not need to know ID of receiver but they must exist at the same time</td>
<td>e.g. IP multicast</td>
</tr>
</tbody>
</table>
Group communication

- Sender communicates with a group, as a whole, without knowing the identity of members
  - An abstraction over multicast communication

- Group management
  - Processes may join/leave a group, membership service provides interface for membership changes, failure detection and notification
  - Groups may be closed/open, overlapping or non-overlapping and synchronous or asynchronous

- Some systems support reliable and ordered multicast
  - Reliable multicast, besides integrity and validity, agreement – if the msg is delivered to one, is delivered to all
  - Ordered multicast also support FIFO, causal (causally related messages arrive in the same order everywhere) or total ordering (all msgs arrive in the same order everywhere)
Group communication

- **JGroups toolkit**
  - An example based on Birman and van Renesse’ work on ISIS, Horus and Ensemble at Cornell U.
  - Includes channels, building blocks and a composable stack
  - CAUSAL – causal ordering
  - FRAG – configurable packetization
  - MERGE – Network partitions and group merges
  - …
Publish-subscribe

- AKA distributed event-based systems; the most widely used of all indirect communication models

- The pub/sub system job – match subscriptions w/ published events and ensure correct delivery of event notifications

- A wide variety of application domains
  - financial information and other feeds of real-time data
  - support for cooperative working
  - monitoring applications
Publish-subscribe

Some variants of subscription models
- Channel-based – basic, publishing to named channels
- Topic or subject based – notifications are expressed in terms of a number of fields; one field denotes the topic
- Content-based – allows subscription over a range of fields
- Other types explored include type-, context- and concept-based and more complex event processing

Implementation issues
- Goal – efficient delivery of the right events to the right subscribers w/ appropriate security considerations
- Centralized/distributed – centralized event broker, network of brokers or full p2p
- Routing options include flooding, filtering, rendezvous
Architecture of publish-subscribe systems

Event routing
- Flooding
- Filtering
- Rendezvous
- Informed gossip

Overlay network
- Broker network
- Group multicast
- DHT
- Gossip

Network protocols
- TCP/IP
- IP multicast
- 802.11g
- MAC bcast
Message oriented

- Point-to-point comm. through an intermediary queue
- Asynchronous persistent communication through support of middleware-level queues – queues correspond to buffers at communication servers
- Common primitives
  - Put – append a msg to a given queue
  - Get – Block until the specified queue is non-empty and remove the first msg
  - Poll – Check a given queue for message and remove first, never block
  - Notify – Install a handler to be called when a msg is put into the given queue
- Queuing of msgs. can be FIFO or priority-based
- E.g. IBM WebSphere MQ, Java Messaging Service
Message brokers and apps. integration

- Message queuing systems assume a common messaging protocol: all applications agree on message format
- To use MQ systems for integration – message broker takes care of application heterogeneity
  - Transforms incoming messages to target format
  - Often acts as an application gateway
  - May provide subject-based routing capabilities
Shared memory approaches

- **Distributed shared memory**
  - Access appears to be to a shared address space
    - Done by underlying runtime system using msg. passing
  - Reading and writing bytes
  - Accessed by address

- **Tuple space**
  - First introduced with *Linda* by D. Gelernter now adopted by IBM Tspaces, JavaSpaces, etc.
  - Higher-level than DSM; semi-structured data
  - Associative access – write into a tuple space and read or take from it
**Summary**

- The power of indirection in communication – communication through an intermediary
  - Uncoupling in space and/or time

<table>
<thead>
<tr>
<th></th>
<th>Group</th>
<th>Pub/sub</th>
<th>MQ</th>
<th>DSM</th>
<th>Tuples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Space uncoupled</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Time uncoupled</td>
<td>Possible</td>
<td>Possible</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Style</td>
<td>Comm</td>
<td>Comm</td>
<td>Comm</td>
<td>State</td>
<td>State</td>
</tr>
<tr>
<td>Comm pattern</td>
<td>1-m</td>
<td>1-m</td>
<td>1-1</td>
<td>1-m</td>
<td>1-1/1-m</td>
</tr>
<tr>
<td>Scalability</td>
<td>Limited</td>
<td>Possible</td>
<td>Possible</td>
<td>Limited</td>
<td>Limited</td>
</tr>
<tr>
<td>Associative</td>
<td>No</td>
<td>Content-based only</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>