Distributed Coordination-Based Systems

- Coordination Models
- Architectures
- Communication
- Naming
- Synchronization
- Consistency and Replication
- Fault Tolerance
- Security
- Federative Systems

Coordination Models

- Coordination of components
- Transparent distribution of components
- Separate computation and coordination
- Different reference types (in program code):
  - Coupled - directs access
  - Decoupled
- Different timing types (sync):
  - Coupled
  - Decoupled

<table>
<thead>
<tr>
<th>Reference / Temporal</th>
<th>Coupled</th>
<th>Decoupled</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direct</td>
<td>Mailbox</td>
<td></td>
</tr>
<tr>
<td>Moving oriented</td>
<td>Generative communication</td>
<td></td>
</tr>
</tbody>
</table>

Architectures

- Data described by attributes
  - \((\text{attribute}, \text{value})\)
- Read and write by Publish - Subscription
- After write
  - subscription attributes have to be matched against the written data
  - then
    - the values have to be forwarded to the subscribers
    - or
    - the subscribers are notified

Architectures (cont)

- Simplest (and most common)
  - Client-Server
- Peer-to-Peer Architecture
  - for scalability
  - Distributed Hash Table (DHT)
### Communication

- **Local Area Networks**
  - Multicast for data distribution
  - Only subscribed data will be considered by receiver

- **Wide Area Networks**
  - Optimized data distribution
    - Example: using the p-tree from an echo algorithm
    - Each node knows where to send values for different attributes

### Naming

- Subscriptions can be for a vector of attributes
  
  \[(attribute1, value), (attribute2, value), (attribute3, value)\]

- If we have one attribute, *attribute, value*, it is called an **event**.

### Composite Events

- Subscriptions can also be for more complicated behaviour, **Composite Events**

- **Examples:**
  
<table>
<thead>
<tr>
<th>Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>Notify when room R4 is occupied</td>
</tr>
<tr>
<td>S2</td>
<td>Notify when room R4 is unoccupied and the door is unlocked</td>
</tr>
<tr>
<td>S3</td>
<td>Notify when room R4 is unoccupied for 10 seconds while the door is unlocked</td>
</tr>
<tr>
<td>S4</td>
<td>Notify when the temperature in room R4 rises more than 1 degree per 30 minutes</td>
</tr>
<tr>
<td>S5</td>
<td>Notify when the average temperature in room R4 is more than 20 degrees in the past 30 minutes</td>
</tr>
</tbody>
</table>

- Can be described by a Finite State Machine,
  - S3:

  ![Finite State Machine Diagram](chalmers.com)
Synchronization

- Easy when using one single server
- Much more complicated when using shared dataspace and replication

Consistency and Replication

- Copies of data
- To avoid problems:
  - Two-phase-commit

Fault Tolerance

- As by now fault tolerance has not been much addressed by Distributed Coordination Systems!
- One proposed solution:
  - Pragmatic General Multicast, PGM
    - uses sequence numbers for data multicasts
    - when missing a multicast a node requests a resend
    - nodes keep the multicast messages for a while in order to fulfill this wish.

Security

- Information confidentiality
  - Encryption to prevent middleware to read data.
- Subscription confidentiality
  - prevent middleware from knowing what data is subscribed
- Publication confidentiality
  - Encryption to prevent middleware to write data
- Decoupling Publishers from Subscribers
- Secure Shared Dataspaces
  - very little work is done to achieve this
**Federative Systems**

- Very loosely coupled processes
- The processes are not allowed to hinder each other
- No waiting
- Messages through special Middleware

**Middleware Placement**

- Distributed applications
  - Applications
- Distributed facilities
  - Transaction System
  - Electronic mail
- Program groups
  - RPC
  - Database
  - Message handling
  - ORB
- Communication services
  - Middleware
- Basic communication
  - Network software

**Message handling**

- Three types:
  - Not persisting
    - memory based message queues
  - Persisting
    - disc based message queues
  - Transactional
    - no dubious states in message queues
General requirements

- An information system consisting of subsystems communicating using messages.
- Each generated message should correspond to a normal interaction within the company:
  - A limited number of messages will be exchanged between the autonomous subsystems.
  - Reasonable performance requirements.
  - Quite obvious to identify what messages that should be exchanged.

General requirements (cont.)

- High communication independence:
  - Each subsystem should be able to send a message at any time. Receiver state should have no influence.
  - Each subsystem should be able to be restarted at any time without affecting the message system.
  - The interface between subsystems and message handling system should be as transparent as possible.
  - The receiving subsystem should be able to read incoming messages how it finds it desirable.
  - Handling of messages should be done by the receiving subsystem.
  - The message handling system should have no knowledge of the messages content.

- It is also desirable to make the information system able to work even when the message handling system does not.
  - Alternative solutions when the message handling system does not function.

Possible solutions

- Centralized
- Semi distributed
- Distributed

Centralized solution

System 1 (Local network)

| Computer 1 | Application 1 |
| Application 2 |
| Computer 2 | Application 3 |
| Application 4 |

Server 1

TA

AA

System 2 (Local network)

| Computer 3 | Application 5 |
| Application 6 |

Server 2

TA

AA

inter network

AA: Application Agent

TA: Transfer Agent
Centralized solution

- **Advantages**
  - Easy administration.
  - Easy supervision.
  - Possibility for efficiency.

- **Disadvantages**
  - Communication independence will not be so high. The local server must be working.
  - Risk for network dependent applications when there is communication with AA using the local network.
  - Application interfaces might be complicated.

---

Semi distributed solution

- **Advantages**
  - More communication independence. No dependence between applications and the local server.
  - Application interfaces will be more network independent. What is network dependent will be in AA.
  - Application interfaces will be less complicated.

- **Disadvantages**
  - Administration of the message passing system will be harder.
  - The supervision will be slightly more complicated than in a centralized system.
  - Less efficient than a centralized system.
Distributed solution

- Advantages
  - High communication independence.
  - Application interfaces will be even more network independent.
  - Application interfaces will be even less complicated.

- Disadvantages
  - Much harder message passing system administration.
  - Difficult supervision.
  - Much less efficiency.