Coordination-Based Systems

13.1 Coordination Models

Coordination models

Essence
We are trying to separate computation from coordination; coordination deals with all aspects of communication between processes, as well as their cooperation.

Couplings
Make a distinction between
- Temporal coupling: Are cooperating/communicating processes alive at the same time?
- Referential coupling: Do cooperating/communicating processes know each other explicitly?
Coordination-Based Systems

13.1 Coordination Models

Coordination models

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

Architectures: Overview

Essence

- A data item is described by means of attributes.
- When made available, it is said to be published.
- A process interested in reading an item, must provide a subscription: a description of the items it wants.
- Middleware must match published items and subscriptions.

Example: Jini/Javaspaces

Coordination model

Temporal and referential uncoupling by means of JavaSpaces, a tuple-based storage system.
- A tuple is a typed set of references to objects
- Tuples are stored in serialized, that is, marshaled form into a JavaSpace
- To read a tuple, construct a template, with some fields left open
- Match a template against a tuple through a field-by-field comparison
Example: Jini/Javaspaces

Write: A copy of a tuple (tuple instance) is stored in a JavaSpace
Read: A template is compared to tuple instances; the first match returns a tuple instance
Take: A template is compared to tuple instances; the first match returns a tuple instance and removes the matching instance from the JavaSpace

Example: TIB/Rendezvous

Coordination model
Uses of subject-based addressing ⇒ publish-subscribe system.
- Receiving a message on subject X is possible only if the receiver had subscribed to X
- Publishing a message on subject X ⇒ message is sent to all (currently running) subscribers to X.

Example: Lime

Every node has its own dataspace:
- When P and Q are in each other’s proximity, dataspaces become shared
- Published data items are stored locally, until removed
- P can publish data items from specific process
- Reactions describe what to do when a match is found
Content-based routing

Observation
When a coordination-based system is built across a wide-area network, we need an efficient routing mechanism (centralized solutions won’t do).

Solution
- Naive: Broadcast subscriptions to all nodes in the system and let servers prepend destination address when data item is published
- Refinement: Forward subscriptions to all routers and let them compute and install filters.

Content-based routing: naive solution

Replication: Static approaches

Note
Replicating data items to all machines implies broadcasting removals.
Balancing read/write operations

**Problem**
Find a balance between the costs for reads, and writes/removals ⇒ organize dataspace as 2D grid

**Example**
A writes a data item; B wants to read it.

A broadcasts tuple to these machines
B broadcasts template to these machines

Dynamic replication

**Observation: Not all data items are equal**
- Decide on replication on a per-type basis
- Refinement: Let a central component observe read/write patterns and decide on replication strategy (self-replication)