Distributed Systems
Principles and Paradigms

Chapter 12
(version 26th November 2001)

Maarten van Steen

Vrije Universiteit Amsterdam, Faculty of Science
Dept. Mathematics and Computer Science
Room R4.20. Tel: (020) 444 7784
E-mail:steen@cs.vu.nl, URL: www.cs.vu.nl/∼steen/

Distributed Coordination-Based Systems

- Coordination models
- TIB/Rendezvous
- Jini

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.

Make a distinction between:

Temporal coupling: Are cooperating/communicating processes alive at the same time?

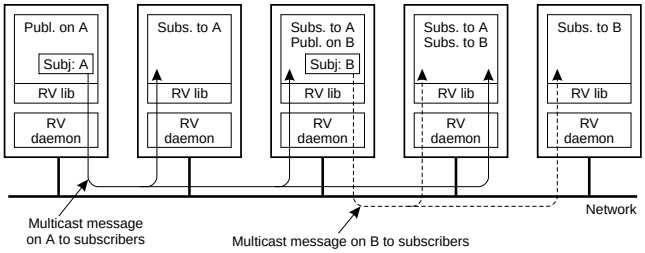
Referential coupling: Do cooperating/communicating processes know each other explicitly?

		Temporal: Coupled	Temporal: Uncoupled
Referential	Coupled	Direct	Mailbox
	Uncoupled	Meeting oriented	Generative communication

TIB/Rendezvous: Overview

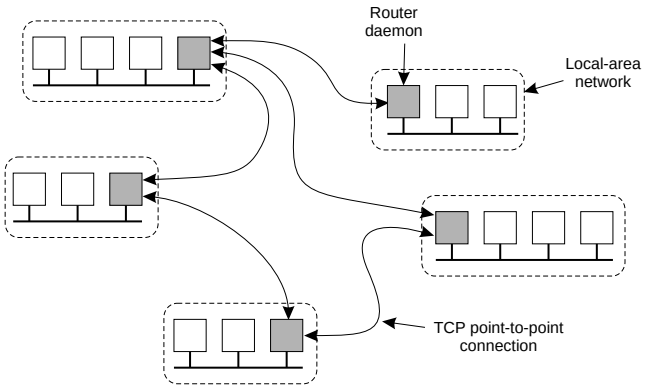
Coordination model: makes use of **subject-based addressing**, leading to what is known as a **publish-subscribe architecture**

- Receiving a message on subject X is possible only if the receiver had **subscribed** to X
- **Publishing** a message on subject X, means that the message is sent to all (currently running) subscribers to X.



Overall Architecture

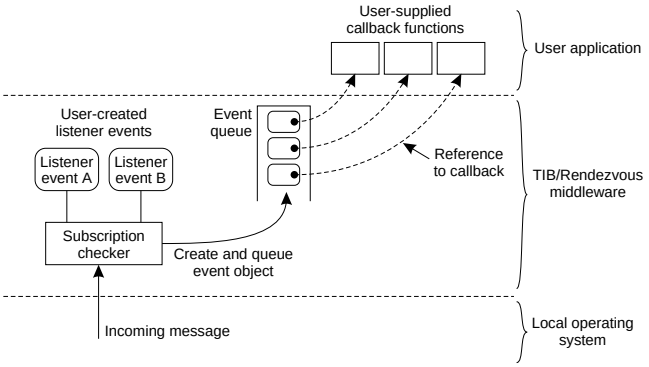
Essence: TIB/RV uses multicasting to forward messages to subscribers. To cross large-scale networks, it effectively builds an **overlay network** with proprietary multicast routers:



Communication: Events (1/2)

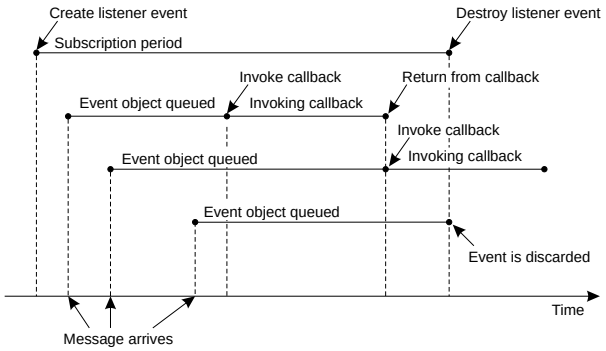
Events: Publish-subscribe systems are ideally supported by means of events: you are notified when someone publishes a message that is of interest to you.

Listener event: local object that registers a **callback** for a specific subject.



Communication: Events (2/2)

Event scheduling: Events for the same listener event are handled one after the other; they may also be lost/ignored if listener event is destroyed at the “wrong” time:



Naming

Essence: Names are important as they form the “address” of a message. Filtering facilities ensure that the right messages reach their subscribers:

Example	Valid?
Books.Comp_Sys.Distr_Sys	Yes
.ftp.cs.vu.nl	No (starts with a “.”)
ftp.cs.vu.nl	Yes
NEWS.res.comp.os	Yes
Maarten..van_Steen	No (empty label)
Maarten.R.van_Steen	Yes

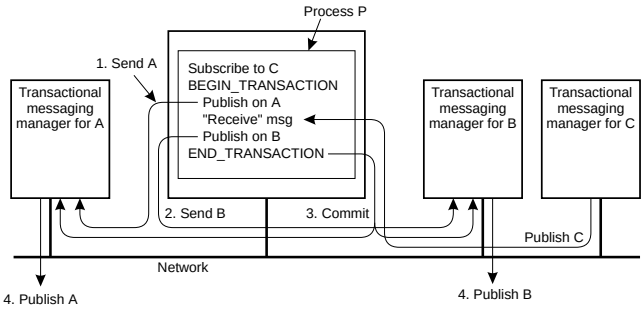
Filtering: using special wildcards:

Subject name	Matches
*.cs.vu.nl	ftp.cs.vu.nl www.cs.vu.nl
nl.vu.>	nl.vu.cs.ftp nl.vu.cs.zephyr nl.vu.few.www
NEWS.comp.*.books	NEWS.comp.os.books NEWS.comp.ai.books NEWS.comp.se.books NEWS.comp.theory.books

Transactional Messaging

Essence: Ensure that the messages sent by a single process are delivered only if the sender commits ⇒ store published messages until commit time, and only then make them available to subscribers

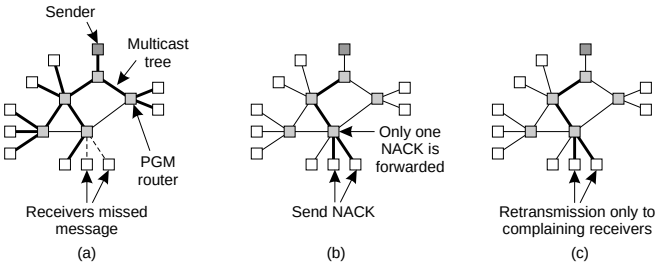
Note: Transactional messaging is not the same as a transaction; only a single process is involved.



Fault Tolerance: Multicasting

Problem: TIB/RV relies on multicasting for publishing messages to all subscribers. This mecahnism needs to be extended to wide-area networks and requires **reliable multicasting**.

Solution: Pragmatic General Multicast (PGM): a NACK-based scheme in which receivers tell the sender that they are missing something (⇒ no hard guarantees).



Fault Tolerance: Process Groups

Essence: Process resilience is provided through process groups; active members respond to all incoming messages, inactive ones just listen.

Note: If number of active members equals one, we have a primary-based replication protocol.

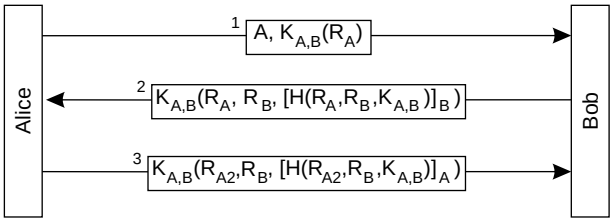
Ranking: All members are ranked; the TIB/RV ensures (automatically) that the highest-ranked process is activated when an active member crashes.

Question: How can the middleware guarantee that a specific number of active members are running?

Security

Essence: Establish a secure channel between a specific publisher and a specific subscriber.

Question: We are losing something in our coordination model – what?



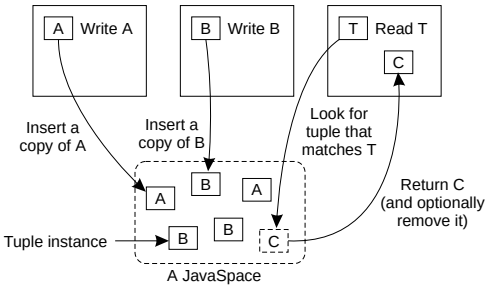
Note: The shared secret $K_{A,B}$ is established through a Diffie-Hellman key exchange. We are now trying to avoid a man-in-the-middle attack (Chuck pretending to be Bob to Alice, and Alice to Bob).

Jini: Overview (1/2)

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

Jini: Overview (2/2)



Write: A copy of a tuple (**tuple instance**) is stored in a JavaSpace

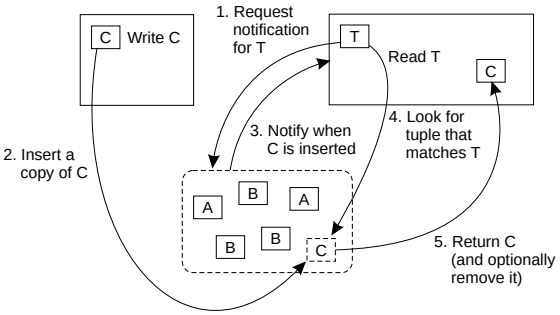
Read: A template is compared to tuple instances; the first match returns a tuple instance

Take: A template is compared to tuple instances; the first match returns a tuple instance and removes the matching instance from the JavaSpace

Communication: Notifications

Essence: A process can register itself at an object to be notified when an event happens. Uses a callback mechanism through **listener objects**. A callback is implemented as an RMI.

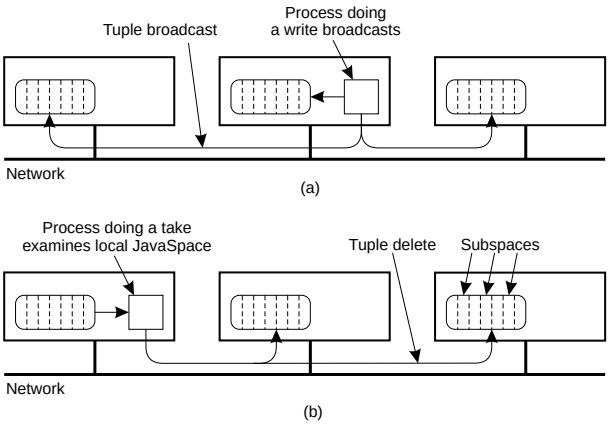
Note: You can also be notified for matches in a JavaSpace, but there may be a race:



JavaSpace Server (1/2)

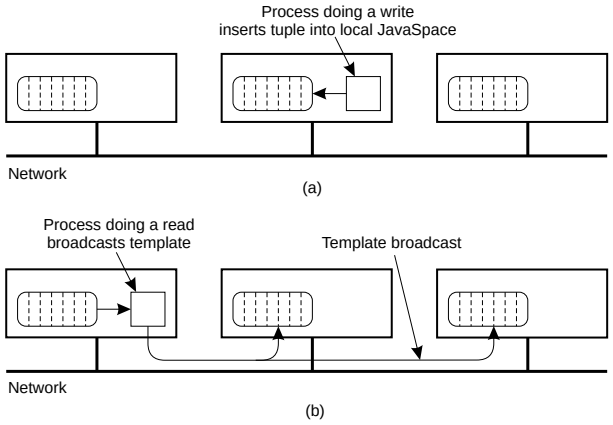
Essence: A JavaSpace is implemented by means of a single server; it turns out be hard to distribute and replicate a JavaSpace.

Replicated version:



JavaSpace Server (2/2)

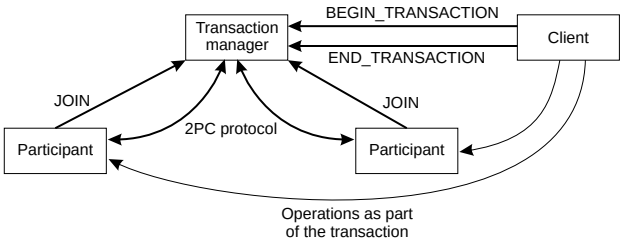
Distributed version:



Scalability: Do not replicate, but use different JavaSpaces leading to nontransparent logical distributions. Possibly move a JavaSpace to places where a lot of clients are.

Transactions

Essence: Jini provides only a standard interface to a 2PC protocol. It offers a default implementation for this protocol.



Question: What good will it do if you only provide interfaces?

Comparison

Issue	TIB/Rendezvous	Jini
Major design goal	Uncoupling of processes	Flexible integration
Coordination model	Publish/subscribe	Generative communication
Network comm.	Multicasting	Java RMI
Messages	Self-describing	Process specific
Event mechanism	For incoming messages	As callback service
Processes	General purpose	General purpose
Names	Character strings	Byte strings
Naming services	None	Lookup service
Trans. (operations)	Messages	Method invocations
Trans. (scope)	Single process	Multiple processes
Locking	No	JavaSpace operations
Caching/replication	No	No
Reliable comm.	Yes	Yes
Process groups	Yes	No
Recovery mech.	No explicit support	No explicit support
Security	Secure channels	Java-based