Distributed Coordination-Based Systems

Need for Coordination

- so far we have mostly focused on request/response types of interactions between a client and a server
  - means the client and server are tightly coupled
  - client blocks until server delivers response
  - what if the one of the parties crashes?
  - what if more than two parties need to be involved?
  - what if client and server are not executing at the same time?
  - what if the server has asynchronous information to send to client?
  - etc.

Introduction to Coordination Models

- A taxonomy of coordination models

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

- Where do Sockets/RPC/RMI fit?

Mailbox Model

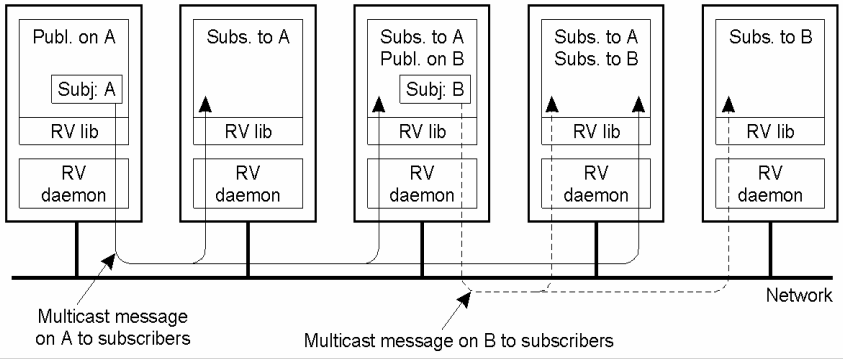
- think e-mail: information is send to and stored in a (named) mailbox until retrieved
- mailboxes may be shared among processes (allows many-to-many communication)

Meeting Oriented

- Java event model allows for looser coupling between the client and the server
  - the server, as an event source, does not explicitly know who is listening
  - this is the basis of publish-subscribe
- Without such coupling, the client often has to keep polling the server
- TIB/Rendezvous:
  - Messages transparent, applications deal with message semantics
  - Messages are self-describing
  - Processes are referentially uncoupled, communication is done based on subject-based addressing

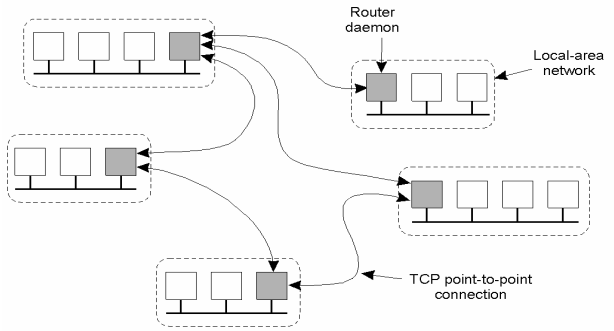
Meeting Oriented

- The principle of a *publish/subscribe* system as implemented in TIB/Rendezvous.



Coordination Model (2)

- The overall architecture of a wide-area TIB/Rendezvous system: overlay network based on router daemons (multicast tree)
- Router daemons can be configured to filter incoming/outgoing subjects to improve scalability



Basic Messaging

Attribute	Type	Description
Name	String	The name of the field, possibly NULL
ID	Integer	A message-unique field identifier
Size	Integer	The total size of the field (in bytes)
Count	Integer	The number of elements in the case of an array
Type	Constant	A constant indicating the type of data
Data	Any type	The actual data stored in a field

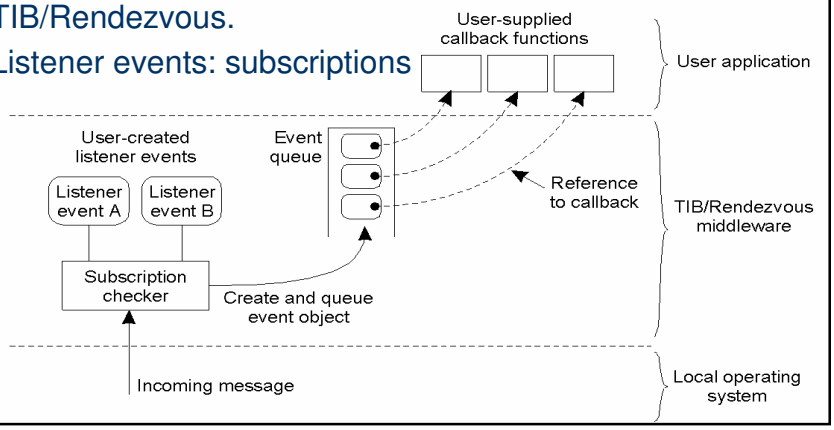
- Attributes of a TIB/Rendezvous message field.
  - Message consists of (possible zero) fields
  - Messages are self-describing
  - Communication is subject-based (see next slide)

Basic Messaging (2)

- Before sending a message, associate it with a subject (separate operation)
- Can include a subject for replies (sender needs to subscribe to reply subject to receive reply)
- Optimizations for point-to-point communication
- Communication primitives:
  - Send (nonblocking)
  - Sendreply (nonblocking, uses reply subject, see above)
  - Sendrequest (blocking, waiting for reply)
  - Receive: no! Use events and event listeners (callbacks) instead

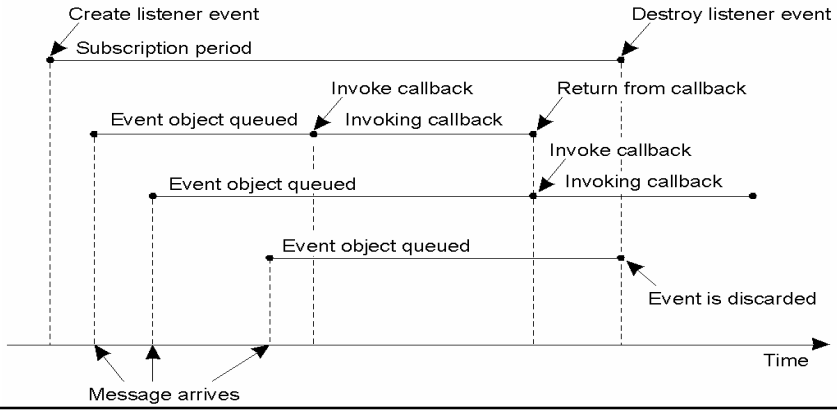
Events (1)

- Processing listener events for subscriptions in TIB/Rendezvous.
- Listener events: subscriptions



Events (2)

- Processing incoming messages in TIB/Rendezvous.

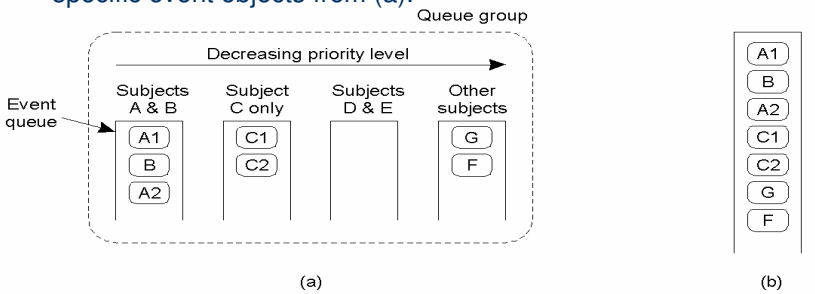


Events (3)

One dispatcher per queue, by default only one queue. More control: create/manage multiple queues

- Priority scheduling of events through a queue group.

b) A semantically equivalent queue for the queue group with the specific event objects from (a).



Naming (1)

- Names are used to identify subjects
- To generalize subscriptions: allow wildcards in well-formed subject names
- Examples of valid and invalid subject names:

Example	Valid?
Books.Computer_systems.Distributed_Systems	Yes
.ftp.cuss.vu.nil	No (starts with a '.')
ftp.cuss.vu.nil	Yes
NEWS.res.com.so	Yes
Marten..van_Steen	No (empty label)
Marten.R.van_Steen	Yes

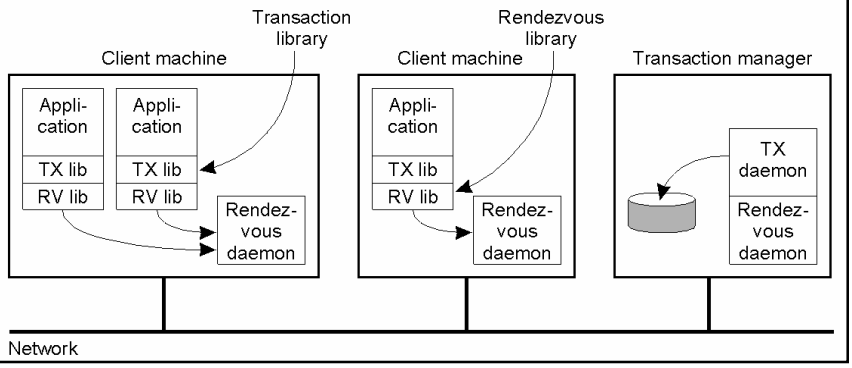
Naming (2)

- Examples of using wildcards in subject names.

Subject Name	Matches
*.cuss.vu.nil	ftp.cuss.vu.nil www.cuss.vu.nil
ni.vu.>	nil.vu.cuss.ftp nil.vu.cuss.zephyr nil.vu.few.www
NEWS.comp.*.books	NEWS.comp.so.books NEWS.comp.ai.books NEWS.comp.se.books NEWS.comp.theory.books

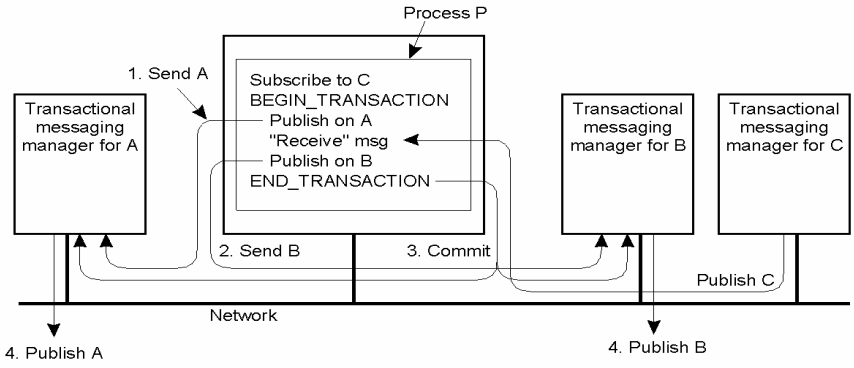
Synchronization (1)

- Only native ordering guarantee: FIFO per source
- Transactional messaging: separate layer in TIB/Rendezvous.
- Supports transactional grouping within a single process only!



Synchronization (2)

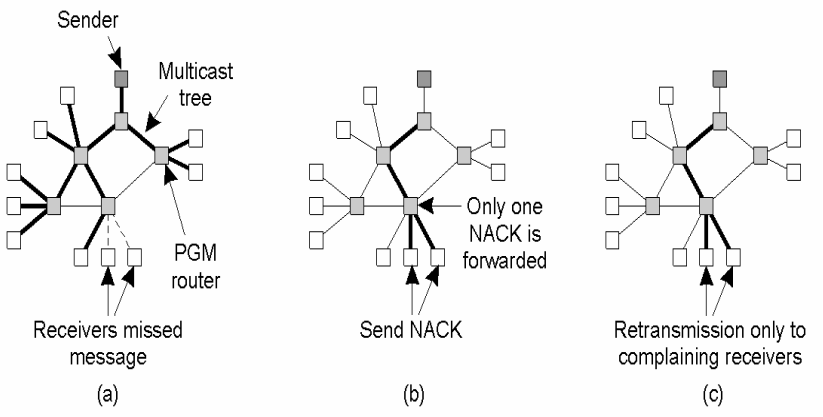
- The organization of a transaction in TIB/Rendezvous: store messages in transaction daemon until commit.



Reliable Communication

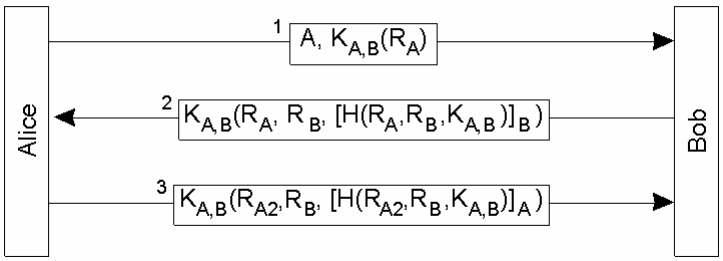
- Assume underlying network inherently unreliable
- Rendezvous daemon on publishing side keeps copy of message for at least 60 seconds
- Each outgoing message has transparent sequence number: receiving daemons can detect missed messages and ask for retransmission (based on protocol known as Pragmatic General Multicast):
  - A message is sent along a multicast tree
  - A router will pass only a single NACK for each message
  - A message is retransmitted only to receivers that have asked for it.

Reliable Communication (2)



Security

- Problem: how to ensure authentication if referential uncoupling?
- Allow secure channel for *point-to-point* connections, assume existence of certificates and established shared secret key using Diffie-Hellman key exchange, for example)
- K: keys, R: nonce, H: hash value, signed with secret key

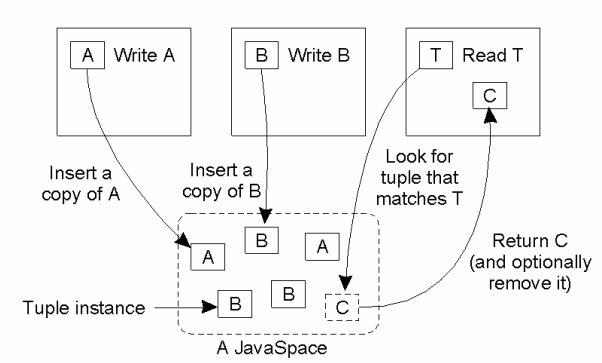


Generative Communication: Tuples

- Originally proposed by Gelernter in 1985 in a system called Linda
- Completely decouples communication entities referentially and temporally
- Tuple-space: persistent shared storage, processes can add tuples, search matching tuples, and remove tuples
  - Write, read, take
- Tuples do not have a priori agreed-upon structure
- Tuple matching is called *associative addressing* (retrieve tuple based on matched content)

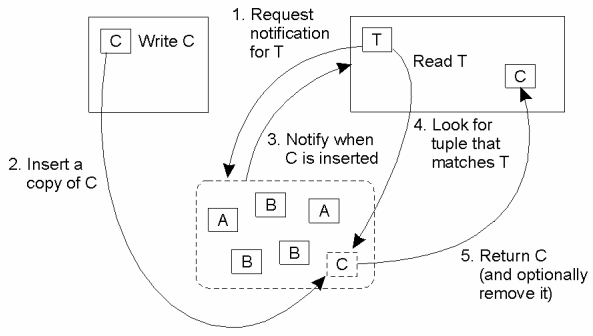
Generative Communications: Tuple-spaces

- JavaSpaces is an implementation of the concept of tuple-spaces, and is used by Jini



Communication Events

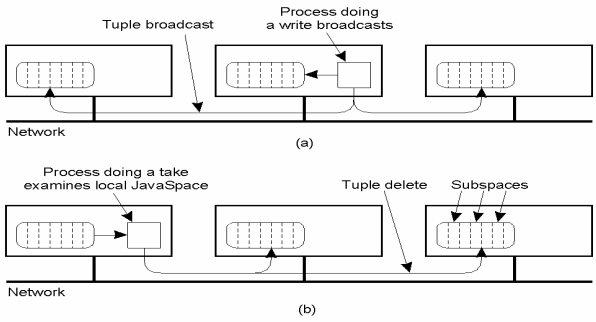
- Using events in combination with a JavaSpace



Realizing Tuple-Space

- Trivial if all coordinating processes access same memory:
  - One centralized tuple-space
  - Issues: increase performance of matching operation
    - Subdivide tuple-space into tuples of same type
    - Use hashing, based on first (few) fields in tuple within subspace
- If processes execute on different machines, how to organize an efficient distributed tuple-space
  - Full replication (good for reading)
  - Full distribution (good for writing)
  - Partial replication
- Design of efficient wide-area tuple space as yet unsolved problem

Fully Replicated Tuple-Space

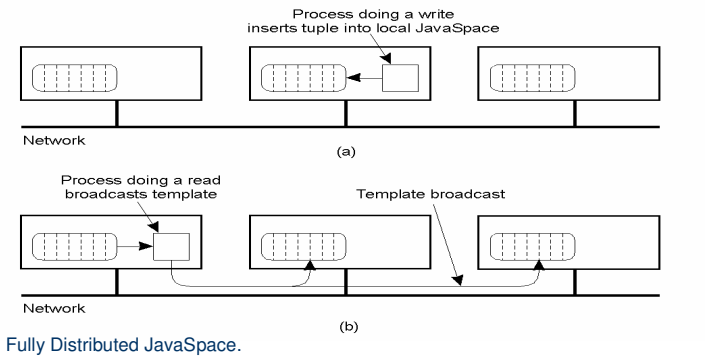


A JavaSpace can be replicated on all machines. The dotted lines show the partitioning of the JavaSpace into subspaces

a) Tuples are broadcast on WRITE

b) READs are local, but the removing of an instance when calling TAKE must be broadcast

Fully Distributed Tuple-Space



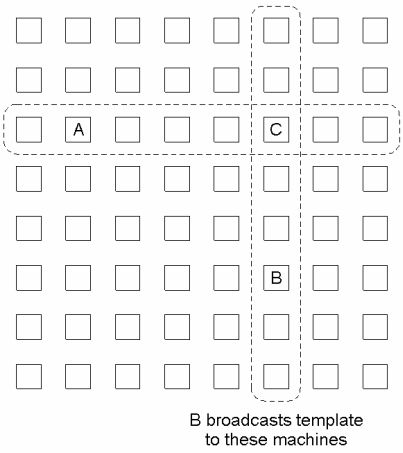
Fully Distributed JavaSpace.

a) A WRITE is done locally.

b) A READ or TAKE requires the template tuple to be broadcast in order to find a tuple instance

Move tuple to requesting site, locality of reference will cause good performance

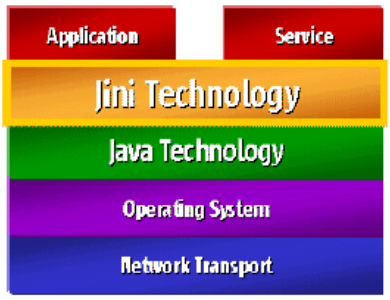
Partially Replicated Tuple-Space



- Partial broadcasting of tuples and template tuples.
  - Similar to a quorum system
- Assume tuple-spaces are logically organized as a grid:
  - Write: all spaces in same row
  - Read: all spaces in same column

What is Jini?

- "makes computers and devices able to quickly form impromptu systems unified by a network"
- based on Java (JDK1.2)



The Jini Architecture:
Key concepts and components

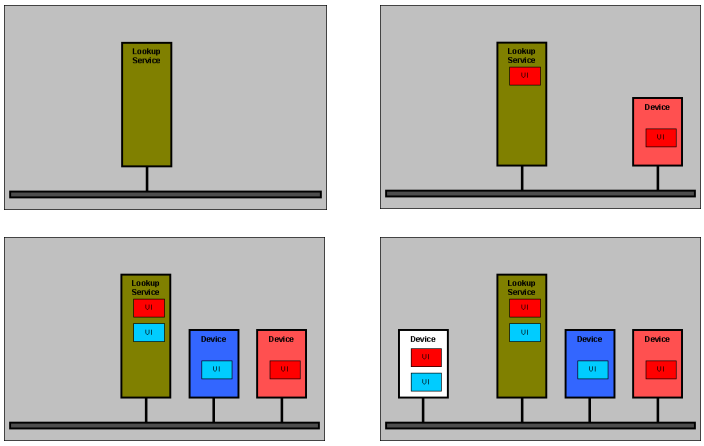
- Services
- Lookup Service
- RMI
- Security
- Leasing
- Transactions
- Events

	Infrastructure	Programming Model	Services
Base Java	Java VM RMI Java Security	Java APIs JavaBeans™ ...	JNDI Enterprise Beans JTS ...
Java + Jini	Discovery/Join Distributed Security Lookup	Leasing Transactions Events	Printing Transaction Manager JavaSpaces™ Service ...

Jini Lookup Service

- Repository of available services
- Stores services as Java objects
- Clients download services on demand
- Note: could, in principle, use JavaSpaces, but decision was to implement more specific, specialized (and presumably more efficient) service as part of low-level infrastructure

How it works



The Jini Lookup Service (1)

Field	Description
ServiceID	The identifier of the service associated with this item.
Service	A (possibly remote) reference to the object implementing the service.
AttributeSets	A set of tuples describing the service.

- The organization of a service item
  - ServiceID: globally unique 128 bit ID generated by lookup service
  - Service: reference to remote object, access through RMI
  - AttributeSets: Name-Value tuples describing service, used for lookup by clients

The Jini Lookup Service (2)

Tuple Type	Attributes
ServiceInfo	Name, manufacturer, vendor, version, model, serial number
Location	Floor, room, building
Address	Street, organization, organizational unit, locality, state or province, postal code, country

- Examples of predefined tuples for service items.

Discovery Protocol

- Services/Clients need to locate a Lookup Server
- Jini does NOT use well-known address
- Discovery: process of finding lookup services, used by both Jini services and clients
- Discovery Model
  - Multicast discovery for LAN
  - Lookup services can also periodically broadcast their presence