CS 194: Distributed Systems
Distributed Coordination-based Systems

Scott Shenker and Ion Stoica
Computer Science Division
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley
Berkeley, CA 94720-1776

Coordination Systems

- Handle all communication and cooperation between processes/objects in a distributed system
  - Emphasize **not** on transparency
  - Object distribution is **explicit**
- Can be classified along two dimensions:
  - **Temporal**: do sender and receiver need to be active simultaneously?
  - **Referential**: do sender need to know the identifier of the receiver?

Taxonomy of Coordination Models

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

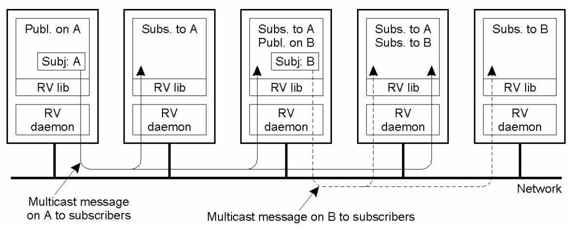
TIB/Rendezvous System

- Meeting oriented model (a.k.a. publish/subscriber)
- Build around concept of **information bus**
- Messages are **subject-based** addressed
  - A message doesn't specify destination, but a **subject name**
- A message is delivered to all objects interested in message's subject

Message Format

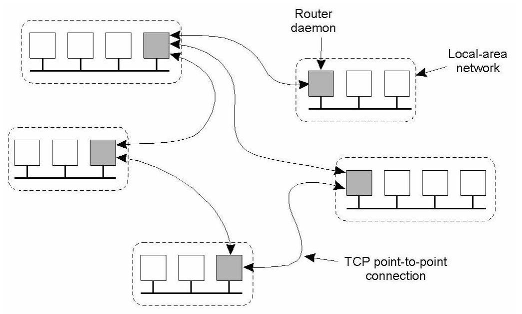
Attribute	Type	Description
Name	String	The name of the field, possibly NULL
ID	Integer	A message-unique field identifier
Size	Integer	The total size of the field (in bytes)
Count	Integer	The number of elements in the case of an array
Type	Constant	A constant indicating the type of data
Data	Any type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

TIB/Rendezvous Architecture



Wide-area Architecture

- Use IP multicast on LANs
- Implement overlay multicast in wide-area



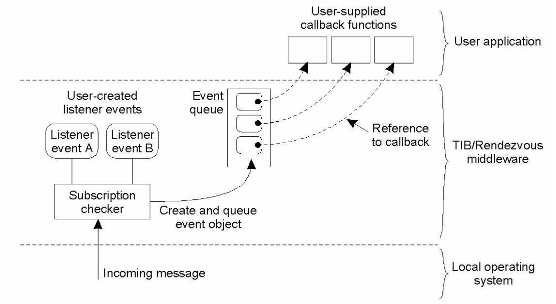
Communication Primitives

- **send()**: send message; non-blocking operation
- **sendreply()**: send a reply upon receiving a message; non-blocking operation
- **sendrequest()**: send message; blocks until a reply is received
- No receive operation; received messages are handled via events

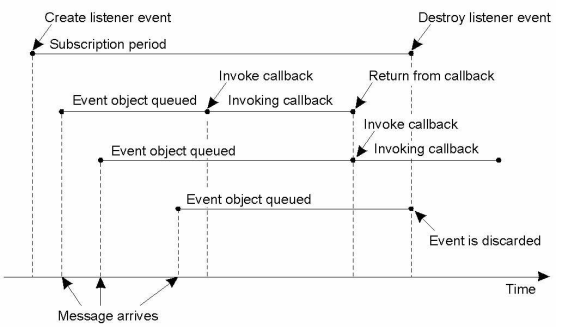
Events

- To subscribe to a subject, create a **listener event** object
- **Listener event** contains reference to a callback function
- When a message arrives, create an **event object** and enque it in an **event queue**
- Each **event queue** is associated a **dispatcher thread**
- **Dispatcher thread** removes object at the head of the queue and invokes callback function

Processing Listener Events

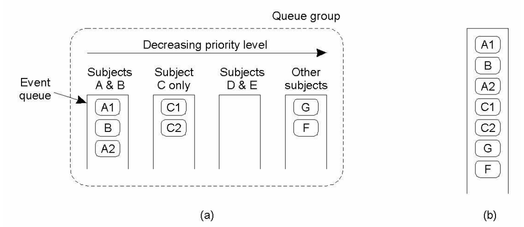


Processing Incoming Messages



Queue Groups

- Assign priorities to event queues



(a) Priority scheduling of events through a queue group
(b) Semantically equivalent queue for the queue group

Naming

- A (subject) name matches a set of sender to a group of receivers
  - Does not identify a resource/object in the system
  - Consists of labels separated by "."

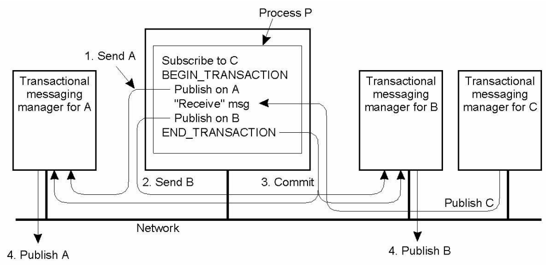
Example	Valid?
Books.Computer_systems.Distributed_Systems	Yes
.ftp.cuss.vu.nil	No (starts with a '.')
ftp.cuss.vu.nil	Yes
NEWS.res.com.so	Yes
Marten..van_Steen	No (empty label)
Marten.R.van_Steen	Yes

Synchronization

- Core of TIB/Rendezvous: FIFO-ordered messages per source
- In addition, transaction messaging: sending/receiving messages can be part of a transaction
  - A separate layer on top of core layer
- A transaction limited to operations that are part of only **one** process
- Transaction manager: stores a message until it has been delivered to all subscribers

Example

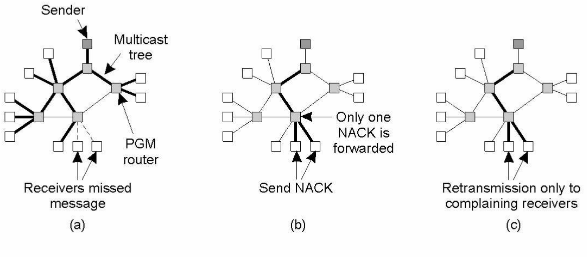
- Process P groups two publish and a receive operations in a transaction
- "Published" messages are sent to corresponding transaction managers
- Messages are published only after transaction is committed



Reliability

- Sending RV daemon
  - assigns a unique sequence number
  - stores it for 60 seconds
- Receiver RV daemon
  - detects whether a message is lost based on sequence numbers
  - request message retransmission
- **Pragmatic General Multicast (PGM)**: scalable implementation of reliability
- Note: this is a "good-enough", not guaranteed reliability

PGM Example



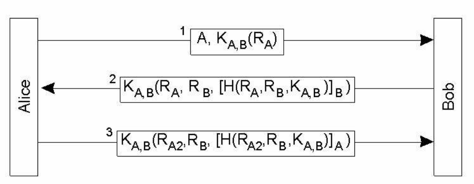
a) A message is sent along a multicast tree
b) A router will pass only a single NACK for each message
c) A message is retransmitted only to receivers that have asked for it.

Security

- Goal: establish a secure channel between a publisher and a subscriber
  - Referential decoupling between publisher and subscriber is lost
- Sender publishes encrypted data including its identity
- Each subscriber sets up a secure channel with the sender
- All subscribers share the same key to decrypt messages

Establishing a Secure Channel

- Diffie-Helman key exchange + public-key cryptography
- Assume Alice and Bob already:
  - obtained certificates containing each-other public key
  - established a shared key $K_{A,B}$ using Diffie-Helman

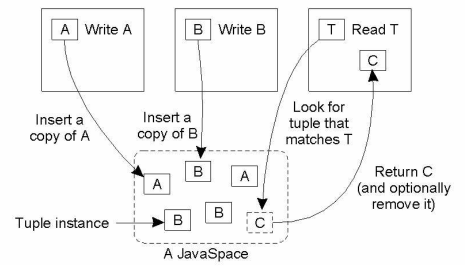

Alice
Bob
1
$A, K_{A,B}(R_A)$
2
$K_{A,B}(R_A, R_B, [H(R_A,R_B,K_{A,B})]_B)$
3
$K_{A,B}(R_{A2},R_B, [H(R_{A2},R_B,K_{A,B})]_A)$

Jini

- Generative communication model
- Built around the concept of tuple space
  - First proposed by Linda
- Tuple space
  - Distributed associative memory
  - Instantiated as a JavaSpace in Jini
- In addition, Jini
  - Provide distributed event and notification system
  - Allow clients discover services when become available

JavaSpace

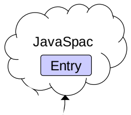
- **write()**: create an object copy and store it in JavaSpace
- **read()**: return tuples from JavaSpace that match a template
- **take()**: like read, but removes tuple from JavaSpace



Example: JavaSpace "Hello World"

```
public class Message implements Entry {
   public String content;
   public Message() {
   }
}
```

```
Message msg = new Message();
Msg.content = “Hello World”;
JavaSpace space = SpaceAccessor.getSpace();
Space.write(msg, null, Lease.FOREVER);
```



Write entry

Example: JavaSpace "Hello World"

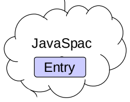
- Use pattern matching to get desired objects from the space
- “null” value represent wildcard
- A message object with the “content” field set to “null” will return any message object
- A message object with the content field set to “Berkeley” will only return a message object with the content set to that value

Example: JavaSpace "Hello World"

```
Message template = new Message();  //Content is null
Message result = (Message)space.read(
       template, null, Long.MAX_VALUE);
System.out.println(result.content);
```

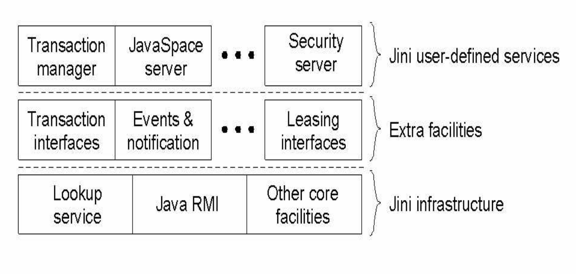
Read entry

“Hello World”



`Long.MAX_VALUE` - timeout parameter

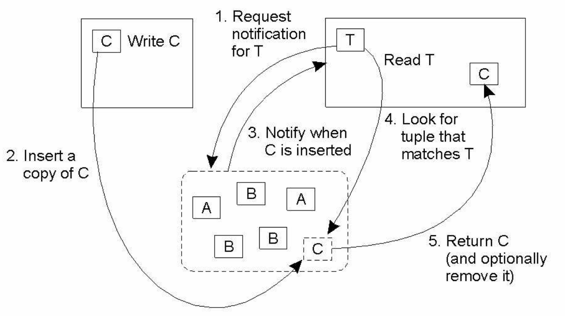
Layered Architecture of Jini



Events

- A client can register with an object that has events of interest
- A client can tell object to pass event to another process
- Notification implemented by remote call

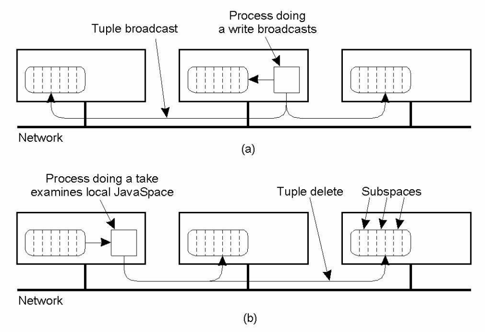
Using Events with JavaSpaces


C
Write C
1. Request notification for T
T
Read T
C
2. Insert a copy of C
3. Notify when C is inserted
4. Look for tuple that matches T
A
B
A
B
B
C
5. Return C (and optionally remove it)

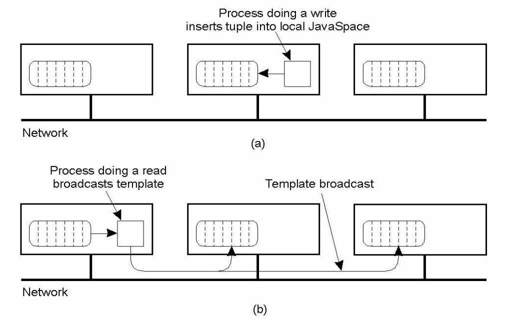
JavaSpace Implementation

- Replicate JavaSpace at all machines
- Store tuples locally, search everywhere
- Partial replication and searching
  - Use DHTs?
- Discussion: what are advantages & disadvantages of these approaches?

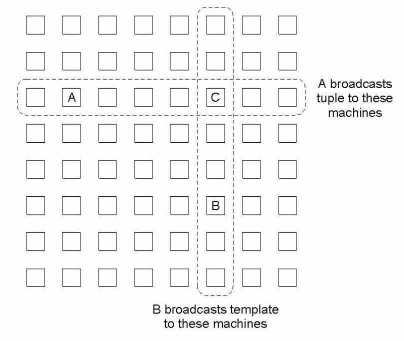
Replicate Everywhere


Tuple broadcast
Process doing a write broadcasts
Network
(a)
Process doing a take examines local JavaSpace
Tuple delete
Subspaces
Network
(b)

Search Everywhere


Process doing a write inserts tuple into local JavaSpace
Network
(a)
Process doing a read broadcasts template
Template broadcast
Network
(b)

Partial Replication and Searching


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

Lookup Service

- Can be implemented using JavaSpaces
  - Each service inserts a tuple describing itself
  - JavaSpace notifies interested clients when service becomes available
- Instead, Jini provides a specialized **lookup service**
  - A service registers itself using (*attribute, value*)-pairs
  - E.g., service parameters, location

Service Item

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

Predefined tuples:

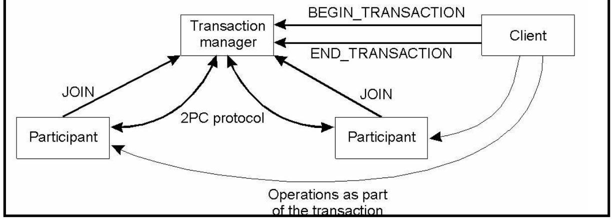
Tuple Type	Attributes
ServiceInfo	Name, manufacturer, vendor, version, model, serial number
Location	Floor, room, building
Address	Street, organization, organizational unit, locality, state or province, postal code, country

Transactions

- Aim to provide ACID properties
  - Atomicity: all operations of a transaction take place, or none of them do
  - Consistency: completion of a transaction must leave the participants in a ``consistent'' state
  - Isolation: activities of one transaction must not affect any other transactions
  - Durability: results of a transaction must be persistent
- Jini
  - Supply the *mechanism* of two-phase commit protocol
  - Leave the *policy* to the participants in a transaction

Transactions

- A transaction is represented by a long identifier, obtained from a transaction manager
- Each transaction is associated a lease; if lease expires, transaction is aborted


Transaction manager
BEGIN_TRANSACTION
END_TRANSACTION
Client
JOIN
JOIN
2PC protocol
Participant
Participant
Operations as part of the transaction