Publish & Subscribe

Larry Rudolph
May 3, 2006
SMA 5508 & MIT 6.883
Agents

- An agent is an autonomous program.
- It executes code and can communicate with other agents.
- All the components in a pervasive computing application (whatever that is) usually called agents
- An agent may be a “proxy” for a device
- Devices, like camera or keyboards, are controlled by some proxy agent
- Agents may appear or disappear at any time
- There is some issue in how to start them
- There can be problems when they crash
- there may be replicates
A collection of agents

‣ Parallel or distributed programming
  ‣ a bunch of communicating agents working to solve a problem
  ‣ faster
    ‣ two heads better than one
  ‣ geographically distributed
    ‣ everyone can’t live together
Agent communication

Two main choices:

- (which was best used to be “religious battle”)

- Shared memory (SM)
  - agents load and store values
  - start with a set of numbers
  - remove two numbers, insert their sum
  - done when only one value remains

- issues: synchronization, locks, etc.

- Message-passing (MP)
Agent communication

Message-passing

- two parts: destination, data

Agent Bob: Send(Alice, “Do you want to go out?”)

Agent Alice: Recv(from,msg)

- from = Bob; msg = “do you want to go out?”

send(Bob, “No”)

Issues:

- Sender must know destination, recv need not blocking or non-blocking
- low performance, lots of copying of data

Note: MP can implement SM and vica-versa

MP on clusters, SM on multiprocessors
Message Passing via Sockets

- Sockets are general
  - Application can specify
    - port
    - protocol
    - other attributes
- Message-Passing
  - library does all the specification
  - may reformat data
Tuple-space

✴ A third communication mechanism!
✴ formed basis of Linda programming language
✴ tuple: ordered collection of typed elements

✴ Basic Operations
✴ **out**: inserts a tuple, whose fields are either
✴ **actual**: a static value
✴ **formal**: a program variable
✴ **in**: extracts tuple, argument is template to match
✴ actuals match fields of equal type and value
✴ formals match fields of same type
✴ **rd**: same as in, but does not remove matched tuple
Tuple-space example

out("jim", 88, 1.5)

{"john", 34, 2.1}
{"mary", 23.1.9}
{"jane", 43.2.0}
{"fred", 56.1.8}

inp("jane", u, v)

rdl("jim", x, y)
Linda programming example

procedure manager
begin
  count = 0
  until end-of-file do
    read datum from file
    OUT("datum",datum)
    count = count + 1
  enddo
  best = 0.0
  for i = 1 to count
    IM("score",value)
    if value > best then best = value
  endfor
  for i = 1 to numworkers
    OUT("datum","stop")
  endfor
end

procedure worker
begin
  IM("datum",datum)
  until datum = "stop" do
    value = compare(datum,target)
    OUT("score",value)
    IM("datum",datum)
  enddo
end
What is the big deal?

- Virtual shared memory
  - tuples with [address, value]
  - stores are inserts, loads are non-destructive reads

- Virtual message passing
  - tuples with [dest, data]
  - recv are destructive reads

- Even more, when matching on multiple fields
  - Allows many types of implementations
Agent Interaction Choices

- Direct communication model
  - Jini
  - FIPA

- Indirect, Shared Data-space models
  - EventHeap (centralized)
  - MARS (fully distributed)

- Event-based publish/subscribe models
  - Siena
  - Jini Distributed Events
  - Selective subscription
Stanford’s Event Heap

- Based on Tuple Space paradigm
- tuple: arbitrary mix of typed fields
- mechanism for passing data & events
- Extensions make it useful for agents
- many projects exist based on different extensions
Event Heap
Extensions

➡ Extended Delivery Semantics:
  ➡ Per-source ordering, always see events in order they are generated by the source
  ➡ Total order: if tuple space is centralized, get this even if multiple sources

➡ Persistent Queries:
  ➡ non-destructive read of those matching
  ➡ also matches tuples inserted in future

➡ Event Notification:
  ➡ like PQ, get notified of future matches
  ➡ at most once semantics
Need more than simple event heap
Suggested additions

✴ Need “distributed, replicated or federated local instances
✴ (from paper by Storz, Friday, & Davies)
✴ Multiple event heap instances -- but not easy of implement
✴ View: processes that share a view have consistent ordering
✴ Session identifiers
✴ non-destructive operation on per-session identifier basis
✴ can share, copy, or destroy id’s for different semantics
More general issues

- Lots and lots of middleware systems
- no winner (may never happen)
- What gets communicated?
- services, events, XML records
- The shared space is often a: BROKER
- The broker stores the tuples and does the matching
Big Issues

➡ Naming
➡ This is a big, big deal.
➡ e.g. how do you name a camera:
➡ model brand, IP, DNS name, location, virtual space
➡ via attributes (color, 740x1024), ownership?
➡ Is there only one name for the agent?

➡ Matching
➡ A big deal
➡ Which attributes explicit, which implicit
➡ Where to do the lookup?
Issues

- Addition information provided by broker
  - for services: how to interface them
  - filtering events
  - higher level events implemented at broker
    - based on multiple basic events

- Adaptivity
  - When to discard services, events
    - keep alive, heartbeats
  - Invoke new instance of service automatically
  - Fault tolerance
Issues

➡ Standards
➡ XML, SOAP, WSDL
➡ Proprietary Interfaces
➡ Middleware may be new Operating System
➡ Whoever controls it will dominate
➡ Not clear if there is or will be a winner
➡ Integration with web-services
➡ Lightweight devices are different
➡ May want stateful communication