Object-Oriented Programming for Wide-Area Computing

Jayadev Misra

Department of Computer Sciences
University of Texas at Austin

Email: misra@cs.utexas.edu
web: http://www.cs.utexas.edu/users/psp

Introduction to Jayadev Misra

On the faculty of the CS department since 1974. (Visiting prof. at Stanford, 1983 -84).

- Teaching

Undergrad course in Discrete Math (CS 336)

Graduate course in Distributed Computing (CS 380D)

Research Seminar in Concurrent Object-Oriented Programming (CS 395T).

Introduction to Jayadev Misra, contd.

- Research Interests

  Distributed Computing

  Algorithms, Languages, Programming Models, Specifications and Proofs, Application domains.

Major Publications

- *Parallel Program Design: A Foundation*, K. Mani Chandy and Jayadev Misra, Addison-Wesley, 1988.
- *A Discipline of Multiprogramming*, Jayadev Misra, Springer-Verlag, 2001. A few chapters are available at `http://www.cs.utexas.edu/users/psp/discipline.ps.gz`

Home page: `http://www.cs.utexas.edu/users/misra`

Wide-Area Computing

For every complex problem, there is a solution that is simple, neat and wrong.

– H. L. Mencken: Prejudices, second series (1920).

Wide-Area Computing: What we do well

- Giant Information Archive

  Navigating and downloading

  Searching

- Point-to-point Communication

  e-mail

  2-party transaction

Wide-Area Computing: What we don't do so well

- Computing with data from a variety of sources

  with no/low human intervention

- Real-time activities

  that triggers a chain of events

  creating alerts, reports, action items

Distributed Applications

- Disaster Recovery
- Flight Cancellation
- Scheduling a Faculty Interview

Core Problem

- Are there a few fundamental concepts?

process, message, communication-channel, caching, consistency, agent, synchronization, exclusive data access, transaction, nested transaction, atomicity, durability, ...

- Can we write real programs with a few concepts?

- Can we evolve the programming model around these few concepts?

Main Thesis

Refine concepts from object-oriented programming, asynchronous concurrent programming, database theory.

A simple model of wide-area computing to:

- Develop applications quickly
- Add new data types, protocols, ..
- Study vulnerability in applications

  Security

  Fault-tolerance
- Prove facts about critical components

A Small Example: Scheduling a Meeting

Schedule a meeting of a set of professors, P, in one of a given set of rooms, R, in the interval $[L, U)$. A meeting can be held at time t if

all professors in P are free at t, and

some room in R is free at t.

Each professor and room has a calendar. For any x:

$x.next(t)$ returns the next time at or after t when x is free.

$x.reserve(t)$ reserves x at t.

$x.next(t) = t$ means x is free at t.

$next$ is ascending, monotonic.

A Program

Given P, R, time interval $[L, U)$ program returns earliest time

t: $L \leq t < U$ if it is possible to meet at t
$t > U$ if it is impossible to meet within $[L, U)$

r: room in which to meet.

Time t is acceptable:

$$\begin{aligned} acc(t) \equiv\ & [t = \langle \max\ p : p \in P : p.next(t) \rangle] \\ & \wedge\ [t = \langle \min\ r : r \in R : r.next(t) \rangle]. \end{aligned}$$

Points to Note

- Multiple instances of this program will interfere.
- Structure of the program:

  professors, rooms are objects

  $next$, $reserve$ are methods
- Some human will invoke this program.

  How about a committee that needs to schedule a meeting every week?

Current Research: Orc

With: Harrick Vin, Young-Ri Choi, Amit Garg

Design a programming model for rapid development of internet-based applications.

- Simple, Simple, Simple
- Strong Theoretical Foundation
- Concurrency for the masses

  easy like spreadsheet

  uses domain expertise

  GUI interface

Faculty Recruiting Problem

1. Call up Ms. X and ask for a preferred date.
2. Call airlines in sequence and reserve.
3. Call hotels in sequence and reserve.
4. Ask the dean for a meeting.
5. Ask the chairman for a meeting.

Faculty Recruiting Problem; contd.

1. Reserve a room for the talk.
2. On completing (4,5 of last page and 1): set up a web sign-up page.
3. Inform A/V people.
4. On completing (2,3 of last page): Inform candidate about flight, hotel.
5. Host, pick-up, lunch/dinner, reimbursement, ...

Forward

Forward message m to addresses e and f, where m, e and f are parameters.

task $forward(message :: m, name :: e, name :: f)\ ()$
  ;*true* $\quad \rightarrow \quad \alpha : send(m, e);\ \beta : send(m, f);\ ()$
end

Non-determinism

Send messages to both e and f, and respond after hearing from any of them.

task $choose(message :: m, name :: e, name :: f)\ name$

;*true* $\rightarrow$ $\alpha : send(m, e); \beta : send(m, f)$

$\alpha(x)$ $\rightarrow$ x

$\beta(x)$ $\rightarrow$ x

end

generalization

A generalization of this problem is as follows. Given a list of names, send messages to all and respond with the name of any respondent.

task $choose(message :: m, [name] :: namelist)\ name$
;*true* $\rightarrow \alpha : send(m, namelist)$
$nonempty.\alpha(x : -) \rightarrow x$
end

Recursion

Send messages to e at 10s intervals until it responds.

task $bombard(message :: m, name :: e)\ ()$

$$
\begin{array}{lll}
;\textit{true} & \rightarrow & \alpha : send(m, e) \\
\alpha & \rightarrow & () \\
\neg\alpha(\mathit{10}\ \text{s}) & \rightarrow & bombard(m, e)
\end{array}
$$

end

Manipulating Lists

Receive a list of names; send mail to each of them and respond only after hearing from all of them.

task $rsvp([name] :: f)\ ()$
    ;*true* $\rightarrow \alpha : send(f)$
    $full.\alpha \rightarrow ()$
end

variation

A variation of this problem is to respond with the list of respondents after receiving a majority of responses.

task $rsvpMajority([name] :: f)\ [name]$
  ;*true* $\rightarrow \alpha : send(f)$
  ; $2 \times |\alpha| \geq |f| \rightarrow \alpha$
end

List pattern Matching

Send a request to names in a list sequentially. Wait for a day to receive a response before sending another request. Respond with the name of the first responder; respond with $Nack$ if there is no responder.

task $hire([name] :: f)\ (Nack \mid Ack\ name)$

$$
\begin{array}{lll}
f([]) & \rightarrow & Nack \\
f(x : -) & \rightarrow & \alpha : send(x) \\
\alpha(y) & \rightarrow & Ack(y) \\
\neg\alpha(1\ \mathrm{d}) \ \wedge\ f(- : xs) & \rightarrow & hire(xs)
\end{array}
$$

end