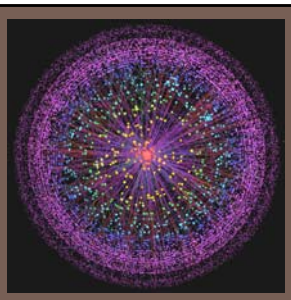


JAVA IN DISTRIBUTED COMPUTING SYSTEMS

Lecture 26 – CS2110 – Fall 2009

Distributed Computing

2

- Up to now we've talked about Java on a single machine
  - Perhaps with threads to exploit multicore parallelism
- But suppose that objects could "live" on other machines?
  - Then if we could just invoke methods on them we would be able to create a distributed program!

Distributed Computing

3

- Java supports this model
  - Called a "Web Services" architecture
  - Your program designates certain interfaces it will make available on the web using *Annotations*

```
package server;
import javax.jws.WebService;

@WebService
public class HelloImpl {
    /**
        * @param name
        * @return Say hello to the person.
        */
        public String sayHello(String name) { return "Hello, " + name + "!"; }
}
```

http://www.artima.com/lejava/articles/threeminutes.html

Talking to the service

4

- Before you can write the client you need to run a program called APT that transforms the server into something that really runs
- APT creates:
  - A so-called "WSDL" file that looks like a web page and describes the new service
  - A"schema" for the messages used to talk to the service
  - Java classes to receive requests and "unpack" them, and to send the response back (which "repacks" them)
  - The client "stub" file

Then…

5

- You start your program on the machine that will be the server
- You also need to wave a magic want to "register" the service with the "Internet Information Service"
- Then on the client machine you import the service and can then write code to talk to it

Talking to our web service

6

- Done using a "client" web-service proxy

```
static void Main(string[] args)
{
    HelloServiceClient proxy = new HelloServiceClient();
    String result = proxy.SayHello("My master");
    Console.WriteLine("Hello Service returned: <" + result + ">");
}
```

- When executed, prints

```
Hello Service returned: <Hello My master!>
```

http://www.artima.com/lejava/articles/threeminutes.html

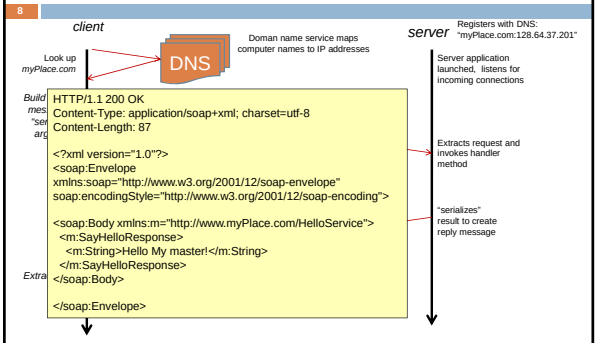
What happened?

7

- A program on one machine invoked an object running on a different machine!
  - You didn't see the code, but the client request was
    - Turned into a message using serialization
    - Sent over HTTP to the service machine
    - Unpacked and then the service method was called
    - Response was serialized back into another message
    - It was sent back to the client machine
- Your client program acted like a web browser!

What happened

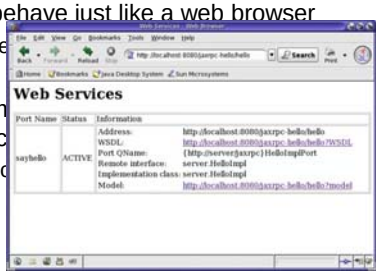
8



Web Browser????

9

- In fact these solutions literally make your client program behave just like a web browser
  - You can e
- And they n
  web site, c
  - And you c



How can an object become a web page?

10

- A web page is just an HTML description of how that page should look
  - HTML is the famous markup language invented by Tim Berners-Lee, a researcher at CERN
  - HTML is actually a "dialect" of XML but we don't need to go there
- Web services use special HTML pages to send requests and make sense of replies

Java serialization

11

- Java has a built in way of taking data in an object and "writing it down" in text format
  - The result looks like a web page
  - It describes the data including types
- Idea is that we can serialize an object, put it into a message, send it to the web service, and get a result
  - Serialized objects are often rather large but the format is extremely general

Magic distributed computing

12

- You can write an object oriented application now but instead of all the objects being on one machine
  - Put them any place you like!
- An object becomes a bit like a web page
- If you know how to find it, you can ask it to do stuff!
  - But must pass arguments by "value", not "reference"

Gotcha's

13

- Reasoning about distributed state is tricky
- Example: the "muddy children" puzzle (Halpern)


"You know the rules! No dessert if you have a muddy face when you come to the dinner table!"

Assumptions

14

- Children don't know if their own faces are muddy… and no child likes to wash his/her face!
- But in fact *every child is muddy*
- Mom repeats herself again and again.
  - Danny reasons: Unless I'm certain my face is muddy, I won't move. But Julia is in BIG trouble! Hee hee hee…
  - Danny (and Julia) don't get dessert

Variation on problem

15

- Same setup but Mom says one more thing: "I see some muddy faces here"
- Then reminds the *n* children *n* times.
  - On n'th repetition, all the children jump up and wash their faces!
- How did they deduce that their faces were dirty?
  - …. You guessed it! **Induction!**

Base case?

16

- Danny is all alone
- Mom says "I see a muddy face here. Better wash up if that face is yours!"
  - (Danny thinks: *I'm the only kid here)*
  - (gulp). "Yes Mommy. I'll do it right now."

N=2

17

- Danny and Julia have muddy faces
- Mom says: "I see a muddy face here. Better wash up if that face is yours!"
  - Danny: *Julia's face is muddy. She's in big trouble!*
  - Julia: *Danny's face is muddy. He won't get dessert!*
    - *… no neither moves*
- *Mom repeats:* "Better wash up if that face is yours!"
  - Danny: *Julia didn't move the first time. If my face had been clean, she would have realized her's was muddy. Ergo my face is muddy!"*
  - Julia reasons identically. Both wash up

N=3

18

- Peter (who hopes his face is clean) looks at Danny, and thinks
  - "*Danny, who also hopes his face is clean, will be looking at my clean face… and at Julia's muddy face and thinking…*
    - "*I see that Danny and Peter have clean faces. Sure home mine is clean too!"*
    - But Julia will realize that Mom's comment ("I see muddy faces") proves that this can't be true
    - Ergo Julia's face is muddy
- Each kid figures this out in round 3.

N large

19

- Assume that the result holds for N-1 children
  - They would all wash their faces on the N-1'st round
- N'th child joins the group
- Can express the same logic we used to reduce from 2 to 1, but now it gets us from N to N-1
- Children all wash up on the N'th round!

Reasoning about distributed systems

20

- Our example reveals that
  - In some ways, these are like other systems. For example, induction is a powerful tool
  - But in other ways they are different
    - Consider Mom. She said "I see some muddy faces" and this somehow made a difference
    - Yet with N>1, children looking around the room could see that *every other child had a muddy face!*
    - So what did Mom tell them that they didn't already know?
- Relates to idea of a "chain of knowledge"
  - She gave them "common knowledge" that someone is muddy

Distributed systems are hard!

21

- Same "problem" posed slightly differently was impossible in one situation, easy in the other
- And issues like this arise all the time
  - In connection to security… privacy… fault-tolerance… consistency

Networking… vs Distributed Computing

22

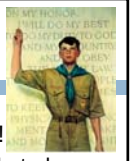
- A "networked" application is one that talks to some resources on some other machine
  - Like a file or a web page
  - Network applications *make no promises.*
- We're used to this "model" and know about its quirks
  - You often get timeouts
  - Sometimes your order is dropped, or goes in twice

Distributed Computing

23

- Some applications (like medical ones) need stronger guarantees:
  - Need to know who the client is
  - And need to "trust" the service
  - May need to protect data against intruders
  - Might want to ensure that the service will be operational even if a crash occurs
- These turn the problem into "distributed computing"

Promises, promises…



24

- A distributed system makes promises!
  - …. I promise to behave like a non-distributed service that never fails
  - …. I promise you'll never notice effects of concurrency
  - …. I won't reveal data to the wrong people. Really!
  - …. Even evil-doers won't stop me from doing the right thing, all the time

Example problem

25

- A hospital has five servers
  - They hold medical record "objects"
  - And we want fault-tolerance
- You write an application to let a doctor enter a new medication order
  - "Put this patient on 2 units of Caldolor per hour"
  - Need to update the servers
- What if something crashes?

Leads to the idea of a "transaction"

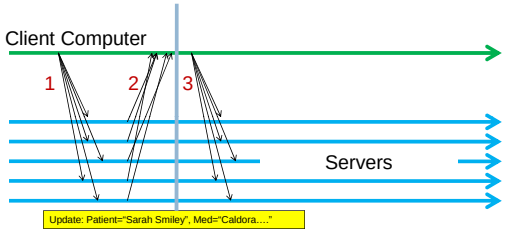
26

- Idea dates to early work on databases
  - Key concept is that either the operation is done to completion, or it fails and does nothing at all
  - A transaction, by definition, must be
    - **atomic**,
    - **consistent**,
    - **isolated**, and
    - **durable**

- How can a client perform an ACID update?

Two-phase commit

27

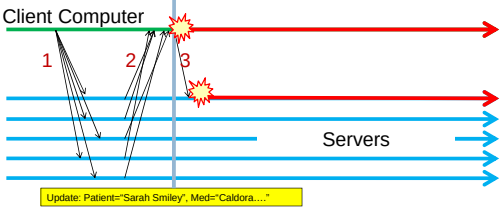


- Idea is to have a "prepare" phase (1, 2) and then a "commit or abort" phase (3)

Problem with two-phase commit

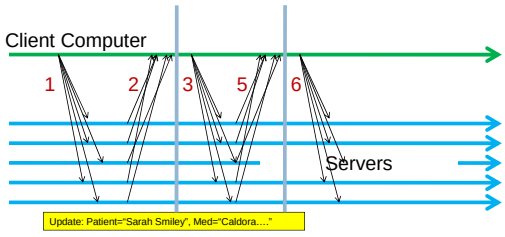
28

- Suppose the client and one machine crash
  - But client had just enough time to send *one* stage-3 msg
- The remainder of the servers might be wedged!



Three-phase commit

29



- With three phase commit can fault-tolerantly do an ACID update… IF failures are detectable

But in general, failures aren't accurately discoverable

30

- It can be shown that any networked system in which crashes can't be "accurately" detected can wedge!
  - Network outage can mimic a machine crashing
  - But famous proof ("Fischer, Lynch and Patterson") shows that in any system where we need to respect what a faulty node thought was true will sometimes hang
- There is a way to evade this "FLP" impossibility result and it relates back to our muddy-kids example
  - Mom needs to tell us which machines failed
  - More generally, we need an "accurate source of knowledge" about crashes

Distributed computing toolkits

31

- Because these problems do get complicated, one area of research is concerned with
  - Solving them well, just once
  - Coding solution as a library that others can use
  - Developers trust the library
- Library can offer fancier functionality
  - Like Mom's Magic Failure Detector!

Fancier problems

32

- These are just two examples from a very interesting research area
  - There are other ways to solve these problems
  - Extending notions of correctness to work with fault-tolerance and concurrency can be a challenge
  - Some researchers argue for solutions that can even guarantee correct behavior under attack!
    - For example, if some service is corrupted and "lies"

Distributed Systems Summary

33

- Basic idea is to treat computers as "homes" where "objects" live
  - Then can do method invocation on objects just by having a URL for them, like a web page
  - But this only yields "networked" applications
  - Biggest issue is that failures are hard to pin down
- Stronger guarantees require "distributed computing" solutions, and get tricky, but can promise things like security, fault-tolerance, consistency…
  - Learn more in classes like cs5410, cs5310, cs6410