SE 550: Lecture 1

Overview

Course summary

Administrivia

Introduction to distributed programming

Example

Summary

Course summary

Fundamentals and techniques of developing distributed object-oriented applications, using a patterns-based approach.

Concepts covered include: networks, client-server architectures, dataflow networks, sockets, message-passing systems, serialization and remote method invocation.

Class structure

1. *Network programming*: the net package.
2. *Streams programming*: the io package.
3. *Protocol specification*: EBNF grammars.
4. *Protocol implementation*: parser generators.
5. *Object serialization*: object streams, SOAP.
6. *Midterm exam*.
7. *RMI 1*: introduction.
8. *RMI 2*: advanced topics.
9. *RMI 3*: implementation.
10. *Object persistency*: long-term state.
11. *Final exam*

Aims

Develop an understanding of distributed programming.

Learn techniques for developing distributed software.

Discover how distributed tools work ‘under the hood’.

Use patterns and OO to formalize solutions.

Objectives

See how to program in a patterns-based style in Java.

Learn the Java tools for distributed programming.

Understand how distributed architectures are implemented.

Administrivia: contact details

Lecturer: Alan Jeffrey

Email: ajeffrey@cs.depaul.edu

Office: CST 840

Phone: (312) 362 8322

Office hours: 3.30-5.00pm Tuesdays and Thursdays.

Administrivia: reading materials

Course home page: http://fpl.cs.depaul.edu/ajeffrey/se550/, contains lectures, homeworks, pointers to API documentation, sample source code...

Textbooks

Java Network Programming by Elliotte Rusty Harold, O'Reilly, 2nd edition, 2000.

Java RMI by William Grosso, O'Reilly, 2001.

Administrivia: prerequisites

Required:

SE 450: Object-Oriented Software Development

CSC 416: Foundations of Computer Science II

Wouldn't hurt:

DS 420: Foundations of Distributed Systems

SE 430: Object-Oriented Modeling

SE 552: Concurrent Software Development

Administrivia: required software

A Java 1.3–1.4 compiler, e.g. Sun's JDK.

An editor for Java source, e.g. XEmacs.

Pointers to software are on the course home page.

Administrivia: assessment

Mid-term exam (10 Feb 2004): 25%

Final exam (16 Mar 2004): 25%

Weekly homeworks (submitted using Courses OnLine, best 7 out of 8): 50%

All students must attend the mid-term and final exams

Late assignments will not be accepted without medical evidence.

Plagiarism or collusion is unacceptable, and will earn an F in the course.

Introduction to distributed programming

What is distributed programming?

What is the difference between distributed programming, parallel programming, and concurrent programming?

What is the difference between a host, a processor, and a thread?

Why distributed programming?

What is object oriented programming?

Why object oriented programming?

Why Java?

Introduction to distributed programming

What is a network? A router?

What is IP? An IP address? A host name? How do hostnames get matched to IP addresses? What is special about addresses 127.0.0.*? 192.168.0.*?

What is a firewall? A proxy? Network Address Translation?

What is TCP? What is a lossy network? What is an unordered network? What is a port? A socket?

What is the client-server model? What is the difference between the client and the server? What are example client/server pairs? Are there other models?

Client example: a simple HTTP client

Download the src.zip archive and unpack it.

Compile and run the HTTP client application with:

```
cd src
javac ajeffrey/teaching/http/client/Main.java
java ajeffrey.teaching.http.client.Main
```

Gotchas:

- Make sure you're in the right directory, and that your PATH and CLASSPATH are set correctly.
- You may need to use backslashes "\" rather than forward slashes "/"

How would we design such an application?

Client example: a simple HTTP client

Built-in classes used by the HTTP client:

- java.net.URL parses URLs
- java.net.Socket creates a socket connection
- java.io.* handles stream-based IO

Hand-built classes used by the HTTP client:

- GUI: the GUI for the application.
- Logic: the business logic for the application.
- SocketIO: contains boilerplate socket code.
- HTTPIO: creates an HTTP input stream from a URL.
- JTextAreaIO: connects an output stream to a text area.
- Pipe: connects an input stream and an output stream together.

Plug the last three together to build a browser!

Server example: a simple HTTP server

Compile the HTTP server with:

```
cd src
javac ajeffrey/teaching/http/client/Main.java
```

Go to a directory containing a index.html file, and run the server:

```
java ajeffrey.teaching.http.server.Main
```

Point a web browser at http://localhost:2000/hello.html

Gotchas:

- localhost may not be recognized: try 127.0.0.1.
- 127.0.0.1 may not work either, try ping 127.0.0.1. If this fails, try installing a dummy network driver!
- Make sure your CLASSPATH is correct, and that the hello.html file is ready to be served.

Server example: a simple HTTP server

Additional built-in classes used by the HTTP server:

- java.net.ServerSocket waits for a socket connection

Additional hand-built classes used by the HTTP server:

- ConnectionManager: manages all of the client connections.
- Connection: manages an individual client connection.

Note: this server took less than a day to write!

Summary

In Java, we can write OO distributed applications using the java.net and java.io API.

Next week: stream-based I/O.

After that: the APIs in more detail.

Homework: Sheet 1, due before next lecture.

Reading: Harold Chapters 4, 10 and 11. Sun's API javadoc documentation on the Socket and ServerSocket classes.