**CS 580 Client-Server Programming**
**Fall Semester, 2002**
**Doc 26 Object Databases, Distributed Objects, WWW**

**Contents**



**References**

Getting Started Using RMI, Old Sun Tutorial on RMI that is no longer on the Web

Integrating FastCGI with Java, http://www.fastcgi.com/devkit/doc/fcgi-java.htm

Philip and Alex's Guide to Web Publishing http://www.arsdigita.com/books/panda/

Apache Modules http://modules.apache.org/

Servlets http://java.sun.com/products/servlet/2.2/javadoc/index.html

Tomcat http://jakarta.apache.org/

JRun http://www.allaire.com/products/jrun/index.cfm

FastCGI http://www.fastcgi.com/

The Common Gateway Interface http://hoohoo.ncsa.uiuc.edu/cgi/overview.html

# Object Databases

Store and retrieve objects from the database

Examples
- Gemstone (http://www.gemstone.com/)
- Objectivity (http://www.objectivity.com/)

No need to convert between objects and sql

# Example – OmniBase for VW

## Creating the database

```
reservations := OmniBase openOn: 'reservationsDB'.
addRoom :=reservations newTransaction.
addRoom root
   at: 'ba121'
   put: (OrderedCollection new).
addRoom commit.
reservations close.
```

## Adding a Request

```
newRequest :=Reservation
   startTime: (Time readFrom: '9:00 am' readStream)
   endTime: (Time readFrom: '10:00 am' readStream)
   date: (Date readFrom: '12/7/02' readStream).

reservations := OmniBase openOn: 'reservationsDB'.
addReservation := reservations newTransaction.
[currentReservations := OmniBase root at: 'ba121'.
currentReservations
   detect: [:each | each intersects: newRequest]
   ifNone:
      [currentReservations add: newRequest.
      currentReservations markDirty]]
         evaluateAndCommitIn: reservations newTransaction
```

# Distributed Objects

- RMI
- Corba
- SOAP
- VW Opentalk

Don’t deal with sockets - make remote method calls

# RMI Example - Hello World

Modified from "Getting Started Using RMI"

## The Remote Interface

```
public interface Hello extends java.rmi.Remote
  {
  String sayHello() throws java.rmi.RemoteException;
  }
```

# The Server Implementation

```
// Required for Remote Implementation
import java.rmi.*;
import java.rmi.server.UnicastRemoteObject;

// Used in method getUnixHostName
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class HelloServer
      extends UnicastRemoteObject
      implements Hello
   {

   public HelloServer() throws RemoteException
      {
      }

   // The actual remote sayHello
   public String sayHello() throws RemoteException
      {
      return  "Hello World from " + getUnixHostName();
      }
```

## // Works only on UNIX machines

```
protected String getUnixHostName()
  {
  try
    {
    Process  hostName;
    BufferedReader  answer;

    hostName = Runtime.getRuntime().exec( "hostname" );
    answer = new BufferedReader(
                new InputStreamReader(
                  hostName.getInputStream()) );

    hostName.waitFor();
    return answer.readLine().trim();
    }
  catch (Exception noName)
    {
    return "Nameless";
    }
  }
```

## // Main that registers with Server with Registry

```
public static void main(String args[])
  {
  // Create and install a security manager
  System.setSecurityManager(new RMISecurityManager());

  try
    {
    HelloServer serverObject = new HelloServer ();

    Naming.rebind("//roswell.sdsu.edu/HelloServer",
                  serverObject );

    System.out.println("HelloServer bound in registry");

    }
  catch (Exception error)
    {
    System.out.println("HelloServer err: ");
    error.printStackTrace();
    }
  }
}
```

# The Client Code

```
import java.rmi.*;
import java.net.MalformedURLException;

public class HelloClient
   {
   public static void main(String args[])
      {
      try {
         Hello remote = (Hello) Naming.lookup(
                              "//roswell.sdsu.edu/HelloServer");

         String message = remote.sayHello();
         System.out.println( message );
         }
      catch ( Exception error)
         {
         error.printStackTrace();
         }
      }
   }
```

Note there are four different types of exception thrown, but shown to save space

# WWW as a Client-Server System

Many people use WWW to implement client-server systems

Advantages:

- WWW browsers are everywhere
- People know how to use WWW
- No need to distribute updates to your client
- Provides links to your services

Disadvantages:

- Browsers have limited functionality
- http is stateless
- http is slow for some applications
- HTML is static

# Generating HTML pages dynamically

WWW client-server systems need more interaction than static web pages

- Dynamically generate HTML pages
- Augment the functionality of browsers

Web server applications

- Code/programs used to dynamically generate HTML

# Main Issues in Dynamically generating HTML

How to map an HTTP request to a program to generate HTML?

Keeping the display separate from the business logic

# Mapping an HTTP request to a program

- Special Web servers
- Special API for Web servers
- Servlets
- CGI
- FastCGI
- Active Web pages (ASP, JSP, SSP, PHP, etc)

# Special Web Servers

Basic HTTP is not hard to implement

Write your own server to produce dynamic responses

Not common

Examples

- Class Wiki

# Special API for Web servers

Web servers allow you to link code to extend their functionality

Examples

- Netscape Server API (NSAPI)
- Internet Server API (ISAPI) - Microsoft
- Apache Modules http://modules.apache.org/
- Servlets

Pros:

- Fast response time

Cons

- Steep learning curve
- Proprietary
- Language dependence
- No process isolation

# Servlets
# Background

Started by Sun

Sun controls the Servlet API
(http://java.sun.com/products/servlet/2.2/javadoc/index.html)

Tomcat - Apache module that supports servlets & Java Server pages

JRun - commercial product supporting servlets & Java Server pages

VisualWorks supports the Servlet API

## What are Servlets?

Java programs that are
- Run as part of a web server
- Mapped to URLs

Servlet API contains 40 classes & interfaces

# Sample Servlet Program

```
import java.io.*;
import java.util.Date;
import javax.servlet.*;

public class SimpleDateServlet extends GenericServlet
  {
  public void service(ServletRequest request,
    ServletResponse response)
    throws ServletException, IOException
    {
    InputStream rawIn;
    DataInputStream  parsedInput;

    rawIn = request.getInputStream();
    parsedInput = new DataInputStream(rawIn);

    ServletOutputStream out = response.getOutputStream();

    String inputLine = parsedInput.readLine();

    if (inputLine.startsWith("date"))
      {
      Date now = new Date();
      out.println( now.toString() );
      }
    else
      out.println( "Invalid Input" );
    }
}
```

# Common Gateway Interface (CGI)

Standard for calling external programs

The basic idea

Web server receives a CGI request

Web server starts the requested CGI program

Web server passes request data to CGI program

CGI program processes request

CGI program return data to Web server

Web server returns response to web client

Pros
- Simplicity
- Language independence
- Process isolation

Cons

- Start up costs for CGI program per request

## The basic questions

- How does the Web server find and run the CGI program?
- How do the Web server and CGI program communicate?
- What information can be sent from Web Server to the CGI program?
- What information can be sent from the CGI program to the Web Server?

## Finding and running the CGI program

Web server is configured to treat files in particular directories as programs

Often cgi-bin indicates a such a CGI directory

All files in a CGI directory and subdirectories are treated as programs

A URL that references a program in a CGI directory will cause the Web server to run that program as a CGI program

## Sending a Response to the server

CGI program writes the response on the standard output

The server reads the output and forms http response

CGI defines headers for communication from CGI program to server

# The CGI Return Headers (CGI/1.1)

The output of scripts begins with a small header. This header consists of text lines, in the same format as an HTTP header, terminated by a blank line (a line with only a linefeed or CR/LF).

Any headers that are not server directives are sent directly back to the client. Currently, this specification defines three server directives:

## Content-type

This is the MIME type of the document you are returning.

## Location

Returns to the server a reference to a document

If is a URL, the server will issue a redirect to the client.

If is a virtual path, the server will retrieve the document specified

## Status

An HTTP/1.0 status line for the server to send to the client.

# CGI Example

**File**: hiMomFancy

**Location**: /net/www/www-eli/cgi-bin/cgiExamples

**CGI directory**: /net/www/www-eli/cgi-bin

**File Contents:**

```
#!/bin/sh

echo Content-type: text/html
echo

echo "<B>Hi Mom</B>",
echo
echo    "<BIG>How is Dad?</BIG>"
echo "<CENTER>I am fine</CENTER>".
```

**URL:**
http://www.eli.sdsu.edu/cgi-bin/cgiExamples/hiMomFancy

## Information from the Web server to the CGI Program

Web server sends data to the CGI Program via:

- Environment variables
- Command line
- Standard input

## Environment Variables

| | | |
|---|---|---|
| AUTH_TYPE | CONTENT_LENGTH | CONTENT_TYPE |
| DOCUMENT_ROOT | GATEWAY_INTERFACE | HTTP_ACCEPT |
| HTTP_USER_AGENT | PATH_INFO | PATH_TRANSLATED |
| QUERY_STRING | REMOTE_ADDR | REMOTE_HOST |
| REMOTE_IDENT | REMOTE_USER | REQUEST_METHOD |
| SCRIPT_NAME | SERVER_NAME | SERVER_PORT |
| SERVER_PROTOCOL | SERVER_SOFTWARE | |

Not all variables are set for each request

See http://hoohoo.ncsa.uiuc.edu/cgi/env.html for more information.

# Example of CGI Environment Variables

**File**: variables

**Location**: /net/www/www-eli/cgi-bin/cgiExamples

**URL**:
http://www.eli.sdsu.edu/cgi-bin/cgiExamples/variables

## File Contents:

```
#!/bin/sh
# Code from http://hoohoo.ncsa.uiuc.edu/cgi/test-cgi.txt
echo Content-type: text/plain
echo

echo CGI/1.0 test script report:
echo

echo argc is $#. argv is "$*".
echo

echo CONTENT_LENGTH = $CONTENT_LE
echo CONTENT_TYPE = $CONTENT_TYPE
echo GATEWAY_INTERFACE = $GATEWAY_INTERFACE
echo HTTP_ACCEPT = "$HTTP_ACCEPT"
echo PATH_INFO = $PATH_INFO
echo PATH_TRANSLATED = $PATH_TRANSLATED
echo QUERY_STRING = $QUERY_STRING
echo REMOTE_ADDR = $REMOTE_ADDR
echo REMOTE_HOST = $REMOTE_HOST
echo REMOTE_USER = $REMOTE_USER
echo REQUEST_METHOD = $REQUEST_METHOD
echo SCRIPT_NAME = $SCRIPT_NAME
echo SERVER_NAME = $SERVER_NAME
echo SERVER_PORT = $SERVER_PORT
echo SERVER_PROTOCOL = $SERVER_PROTOCOL
echo SERVER_SOFTWARE = $SERVER_SOFTWARE
```

# CGI, Java, & Smalltalk

Java and Smalltalk are not good candidates for CGI programs

Expensive to start up VM for each request

Problems reading environment variables

Java's classpath can cause problems

# FastCGI Background

Languages supported:

C, C++, Perl, Java, Python, Smalltalk, TCL

Web servers supported:

Apache, Zeus

Netscape, Microsoft were supported
Third party vendor was bought out, currently not supplying the required modules

## Basic Idea

FastCGI program runs all the time

Web server communicates with FastCGI program via socket

Socket connection stays open

Web server will restart FastCGI program if needed

FastCGI programs can be run as CGI program

Wrapper program available so CGI program can run as FastCGI

# Fast CGI

Pros:

- Fast response time
- Simple
- Multiple language support
- Process isolation
- CGI compatible

Cons

- No Netscape/Microsoft support
- Requires server to know about each fastCGI program

# Active Web Pages
# Basic Idea

Imbed code in html pages

When the web server reads a page the imbedded code is run

## Examples

Java Server Pages (JSP)
Active Server Pages (ASP)
Smalltalk Server Pages (SSP)
PHP

## Pros

Allows graphic designers to design web pages with dynamic content

Scripting languages designed for text manipulation

## Cons

Testing

Some scripting languages are owned by one company

Can result in a code mess with business logic and display code mixed together

# Smalltalk Server Page Example

```
<html>
  <head>
    <title>
      Smalltalk Server Page Example
    </title>
  </head>
  <body>
    <% theHour := Time now hours. %>
    <%= theHour > 18
      ifTrue: ['Good evening.']
      ifFalse: [theHour > 12
        ifTrue:['Good afternoon.']
        ifFalse: ['Good morning.']]. %>
  </body>
</html>
```