

La WWW como Plataforma de Aplicaciones

Dr. Alejandro Zunino
ISISTAN – UNICEN

1

Por qué la WWW es ventajosa?

- Funciona sobre una base mundial ya que sus protocolos se basan en **estándares abiertos** implementados por muchos vendedores en una variedad de máquinas
- La estructura de software para la Web es estrictamente no - propietario, mientras que piezas propietarias pueden acomodarse donde sea necesario
- **También es adecuada para la implementación de aplicaciones distribuidas entre máquinas y redes heterogéneas**

3

Una vista a alto nivel de la WWW

- Los servidores Web proveen acceso a una colección de archivos conteniendo información **hiperenlazada**:
 - el servicio primario es enviar archivos de texto, imágenes, video digitalizado
 - también proveer servicios personalizados a través de interfaces de formularios/scripts CGI
- Los navegadores proveen una interfase gráfica fácil de usar para usuarios que requieren información
- Se mantiene un interfase simple para que se puede ejecutar en todas las redes y la mayoría de las máquinas

5

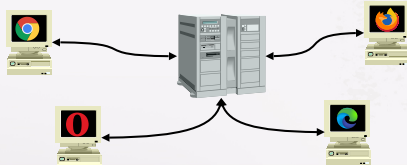
La Arquitectura de la World Wide Web

- La WWW es una colección **hiperenlazada de documentos y programas** que residen en computadoras alrededor del mundo, enlazadas a la **Internet**
- Internet: una colección de protocolos:
 - TCP/IP, DNS, ARP, etc.
 - FTP, SMTP, **HTTP**, Gopher, WAIS, POP, etc.
- Modelo Cliente/Servidor

2

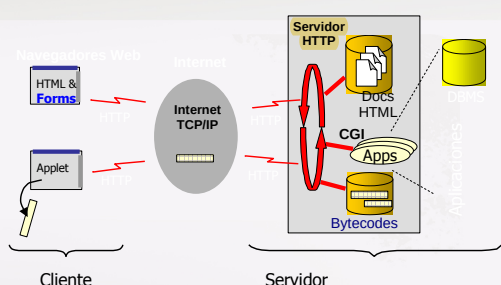
Una vista a alto nivel de la WWW

- Los usuarios pueden usar navegadores WEB para acceder a la información almacenada en los servidores WEB
- Los programas son accesibles a través de Common Gateway Interface (CGI)
- Los clientes envían requerimientos HTTP
- Los servidores responden con documentos (**HTML**, GIF, WAV, etc.)



4

Una vista a alto nivel de la WWW



6

HTTP - HyperText Transfer Protocol

- Protocolo de nivel de aplicación (sobre TCP)
- Sistemas de información hiper medial
- Puede ser considerado:
 - Transferencia de archivos
 - Vínculos entre documentos (**URL**)
 - Transferencia/codificación de hiper medios

Solicitud/Respuesta

- Sin estado en el servidor:

- **No persistente:** analiza req., responde y cierra
- Persistente: analiza req., responde, analiza req., ...

- Para ver una respuesta HTTP:

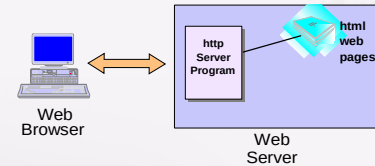
telnet www.exa.unicen.edu.ar 80
GET /isistan/index.html HTTP/1.1 ↔↔↔



7

La Evolución de la Web: Etapa 1

- Sistemas de información
- Contenido estático
- No apto para 'aplicaciones'



9

Ejemplos de Cabeceras HTTP

- en la solicitud:
 - Accept: text/plain, text/html
 - Accept: audio/*
 - Accept-Encoding: x-compress; x-zip
 - UserAgent: Mozilla/5.0 (X11; Linux x86_64; rv:87.0) Gecko/20100101 Firefox/87.0
- en la respuesta:
 - Last-Modified: date
 - Content-Language: en
 - Content-Encoding: x-zip
- en ambos:
 - Content-Length: int

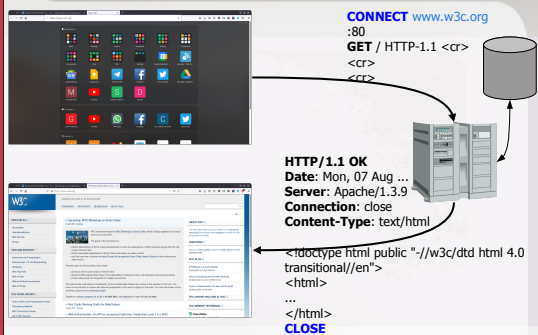
11

URI - Uniform Resource Identifier

- Identifica recursos
 - Nombre (nombre de archivo, número de tabla/registro)
 - Lugar (dir. IP, nombre sitio, puerto)
 - Protocolo (http, ftp, https, gopher)
 - Extras
- HTTP:
 - http_URL = "http://" <host> [":" <port>] [<abs_path> ["?" <query>]]
- Ejemplos:
 - http://www.w3.org/Protocols/
 - http://www.google.com/search?q=covid+stats
 - https://localhost:8080/webmail

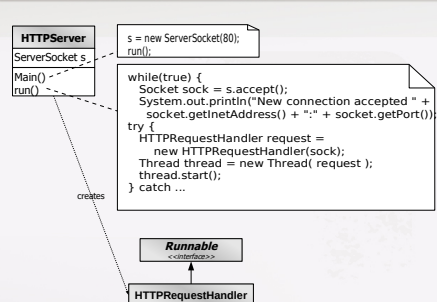
8

Solicitando un Recurso



10

Un Servidor HTTP en Java



12

Un Servidor HTTP en Java

```
graph TD
    Runnable[Runnable  
<<Runnable>>] --> HTTPRequestHandler[HTTPRequestHandler]
    subgraph HTTPRequestHandler
        CRLF["final static String CRLF=\"\r\n\""]
        SocketSocket["Socket socket"]
        InputStream["InputStream is"]
        OutputStreamOS["OutputStream os"]
        BufferedReaderBR["BufferedReader br"]
        HTTPRequestHandlerSocketS["HTTPRequestHandler(Socket s)"]
        run["run()"]
        processRequest["processRequest()"]
        sendBytesFileOutputStreamFS["sendBytes(FileOutputStream fs, ...")
        OutputStreamOS2["OutputStream os)"]
        StringContentTypeStringFileName["String contentType(String fileName)"]
    end
    SocketSocket -.-> SocketCode["socket = s;  
is = s.getInputStream();  
os = s.getOutputStream();  
br = new BufferedReader(  
    new InputStreamReader(s));"]
    HTTPRequestHandlerSocketS -.-> ProcessRequestCode["processRequest();"]
    run -.-> ByteBufferCode["byte[] buffer = new byte[1024];  
int bytes = 0;  
while((bytes = fs.read(buffer)) != -1)  
    os.write(os.write(buffer, 0, bytes));"]
    sendBytesFileOutputStreamFS -.-> FileEndCode["if (fileName.endsWith(\".htm\") ||  
    fileName.endsWith(\".html\"))  
    return \"text/html\";  
..."]
```

The diagram illustrates the structure of an HTTP server in Java. It features a **Runnable** interface (represented by a box with a lightning bolt) that the **HTTPRequestHandler** class implements. The **HTTPRequestHandler** class contains several static fields: `final static String CRLF = "\r\n"`, `Socket socket`, `InputStream is`, `OutputStream os`, and `BufferedReader br`. It also includes a constructor `HTTPRequestHandler(Socket s)`, a `run()` method, a `processRequest()` method, a `sendBytes(FileOutputStream fs, ... OutputStream os)` method, and a `String contentType(String fileName)` method. Dashed arrows indicate the flow of data and control between these methods and external code snippets. The `socket` field is used to create `InputStream` and `OutputStream` objects. The `run()` method calls `processRequest()`. The `processRequest()` method reads data from the `InputStream` into a buffer and writes it to the `OutputStream`. The `sendBytes()` method writes data to the `OutputStream` and checks if the file name ends with ".htm" or ".html" to return the appropriate content type.

Runnable
<<Runnable>>

HTTPRequestHandler

final static String CRLF = "\r\n"
Socket socket
InputStream is
OutputStream os
BufferedReader br
HTTPRequestHandler(Socket s)
run()
processRequest()
sendBytes(FileOutputStream fs, ...
OutputStream os)
String contentType(String fileName)

socket = s;
is = s.getInputStream();
os = s.getOutputStream();
br = new BufferedReader(
 new InputStreamReader(s));

processRequest();

byte[] buffer = new byte[1024];
int bytes = 0;
while((bytes = fs.read(buffer)) != -1)
 os.write(os.write(buffer, 0, bytes));

if (fileName.endsWith(".htm") ||
 fileName.endsWith(".html"))
 return "text/html";
...

13

Un Servidor HTTP en Java

```
// Construct the response message.  
String serverLine = "Server: simple java httpServer";  
String statusLine = null;  
String contentTypeLine = null;  
String entityBody = null;  
String contentTypeLengthLine = "error";  
if ( fileExists ) {  
    statusLine = "HTTP/1.0 200 OK" + CRLF ;  
    contentTypeLine = "Content-type: " +  
        contentType( fileName ) + CRLF ;  
    contentTypeLengthLine = "Content-Length: "  
        + ( new Integer(fis.available()).toString() ) + CRLF;  
}  
else {  
    statusLine = "HTTP/1.0 404 Not Found" + CRLF ;  
    contentTypeLine = "text/html" ;  
    entityBody = "<HTML>" +  
        "<HEAD><TITLE>404 Not Found</TITLE></HEAD>" +  
        "<BODY>404 Not Found"  
        + "<br>usage:http://yourHostName:port/"  
        + "fileName.html</BODY></HTML>";  
}
```

15

- HTML es el formato estándar de la WWW
- Puede ser creado y procesado por variedad de herramientas:
 - editores de texto simples
 - editores WYSIWYG
 - automático
- Utiliza tags tales como `<h1>` y `</h1>` para **estructurar** texto en:
 - encabezamientos
 - párrafos
 - listas
 - vínculos
 - tablas

Un Servidor HTTP en Java

```
while(true) {  
    String headerLine = br.readLine();  
    System.out.println(headerLine);  
    if(headerLine.equals(CRLF) || headerLine.equals("")) break;  
  
    StringTokenizer s = new StringTokenizer(headerLine);  
    String temp = s.nextToken();  
  
    if(temp.equals("GET")) {  
        String fileName = s.nextToken();  
        fileName = "." + fileName ;  
  
        // Open the requested file.  
        FileInputStream fis = null ;  
        boolean fileExists = true ;  
        try {  
            fis = new FileInputStream( fileName ) ;  
        }  
        catch ( FileNotFoundException e ) {  
            fileExists = false ;  
        }  
    }  
}
```

14

Un Servidor HTTP en Java

```
// Send the status line.
output.write(statusLine.getBytes());
// Send the server line.
output.write(serverLine.getBytes());
// Send the content type line.
output.write(contentTypeLine.getBytes());
// Send the Content-Length
output.write(contentLengthLine.getBytes());
// Send a blank line to indicate the end of the header lines.
output.write(CRLF.getBytes());
// Send the entity body.
if (fileExists) {
    sendBytes(fis, output);
    fis.close();
} else output.write(entityBody.getBytes());
}
}
try {
    output.close();
    br.close();
    socket.close();
} catch (Exception e) {}
```

16

Un Ejemplo

```
<HTML>

<HEAD>
  <TITLE>My First HTML Document</TITLE>
</HEAD>

<BODY>
  <H1>This is the <EM>first</EM> section</H1>
  <P>This document shows the usage of some <A
    HREF="http://www.w3.org/MarkUp/">HTML</A> tags:</P>
  <OL>
    <LI>the first list item</LI>
    <LI>the second list item
      <UL>
        <LI>first nested item</LI>
        <LI>second nested item</LI>
      </UL>
    </LI>
    <LI>the third list item</LI>
  </OL>
  <A HREF="https://www.isistan.unicon.edu.ar/">IMG
    SRC="https://upload.wikimedia.org/wikipedia/commons/e/e0/Check_green_1_con.svg"
    WIDTH=100></A>
</BODY>

</HTML>
```

18

Un Ejemplo

file:///tmp/index.html

This is the *first* section

This document shows the usage of some [HTML](#) tags:

1. the first list item
2. the second list item
 - first nested item
 - second nested item
3. the third list item



19

Formularios HTML

file:///tmp/form.html

Fill-Out Form Example #3

Godzilla's Pizza -- Internet Delivery Service

Type in your street address:

Type in your phone number:

Which toppings would you like?

1. ☐ Pepperoni.
2. ☐ Sausage.
3. ☐ Anchovies.

To order your pizza, press this button:

21

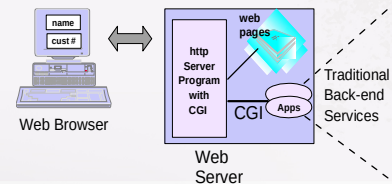
Quando el Usuario Presiona Submit... adiós Web estática

- Los contenidos del formulario son **codificados** y **enviados al servidor de Web**
- Dos codificaciones posibles:
 - **GET**: codificado en el URL para pocos datos
 - **POST**: codificado en el cuerpo de la respuesta HTTP. Para formularios grandes, archivos, datos ocultos, ...
- Ejemplos
 - `http://www.pizza.com/cgi-bin/post-query?`
`address=Campus+Universitario&phone=412345&topping=anchovies`
 - `http://search.netscape.com/cgi-bin/search?search=cgi+reference`
- La consulta es usada para invocar un programa CGI (Common Gateway Interface)

23

La Evolución de la Web: Etapa 2

- Campos, formularios:
 - contenido dinámico
- Procesamiento en el servidor: **CGI**
- Acceso a sistemas legados



20

Formularios HTML

```

<HTML>
<HEAD><TITLE> Fill-Out Form Example #3</TITLE></HEAD>
<BODY>
<H2>Fill-Out Form Example #3</H2>

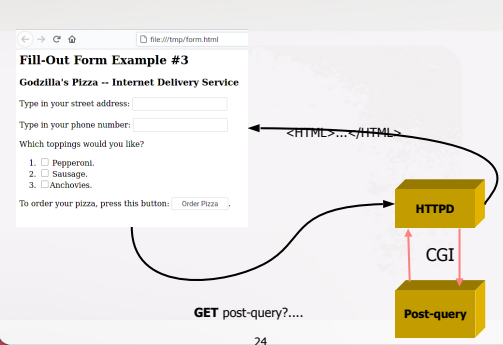
<FORM METHOD="GET" ACTION="http://www.pizza.com/cgi-bin/post-query">
<h3 align="left">Godzilla's Pizza -- Internet Delivery Service</h3>
Type in your street address: <INPUT NAME="address"> <P>
Type in your phone number: <INPUT NAME="phone"> <P>
Which toppings would you like? <P>
<OL>
<LI> <INPUT TYPE="checkbox" NAME="topping" VALUE="pepperoni">
    Pepperoni.
<LI> <INPUT TYPE="checkbox" NAME="topping" VALUE="sausage"> Sausage.
<LI> <INPUT TYPE="checkbox" NAME="topping" VALUE="anchovies"> Anchovies.
</OL>

To order your pizza, press this button: <INPUT TYPE="submit" VALUE="Order
Pizza">. <P>
</FORM>
</BODY>
</HTML>

```

22

CGI - Common Gateway Interface



24

CGI - Common Gateway Interface

- Mecanismo estándar para conectar aplicaciones a servidores HTTP
- Un documento HTML que un servidor solicita es estático
- Un programa CGI es ejecutado para resolver un servicio
- Ejemplo
 - Obtener las transacciones realizadas en una cuenta bancaria
 - Listar las películas de acción del 2003 y sus vínculos eDonkey

25

CGI - Common Gateway Interface (cont.)

- Aplicaciones típicas:
 - Generación dinámica de documentos: conversiones al-vuelo de otros formatos
 - Interfase con otros servicios:
 - bases de datos
 - directorios
 - música por demanda
 - resultados de simulaciones
 - ...
 - Interacciones entre clientes/servidores via Web

27

Un Programa CGI

- Analizar argumentos
 - `http://www.pizza.com/cgi-bin/post-query?address=Campus+Universitario&phone=412345&topping=anchovies`
- Procesar consulta
- **Producir HTML**

```
System.out.print("Content-Type: text/html");
System.out.print("<HTML>");
System.out.print("<HEAD><TITLE>Query
Results</TITLE></HEAD>");
System.out.print("Your address: "+address);
System.out.print("Your phone: "+phone);
...
System.out.print("</HTML>");
```

29

CGI - Common Gateway Interface (cont.)

El cliente para acceder a programas CGI es el mismo que para páginas estáticas!!!!

- Lo único que cambia es el servidor:
 - Cuando el usuario cliquea en un vínculo CGI, el servidor llama al proceso correspondiente con los parámetros del usuario
 - Luego retorna al cliente **la salida del proceso CGI**
 - **NO EL ARCHIVO asociado**

26

Estructura de un Programa CGI

- I Leer la entrada del usuario en los formularios:
 - ésto involucra decodificar los parámetros, datos MIME, etc.
- II Procesar parámetros/datos:
 - por ejemplo consultar una BD
- III Escribir la respuesta en la salida estándar (stdout) en un formato compatible con el navegador: HTML, XML, GIF, PDF, ...

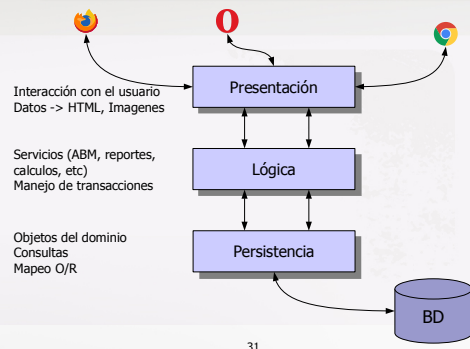
28

Lenguajes para Programación CGI

- Cualquier lenguaje que permita:
 - leer de la entrada estándar
 - escribir en la salida estándar
- De propósito general: C/C++, Java, Python, cualquier Unix shell, etc.
- De propósito específico: Servlets Java, JSP, Frameworks (Struts, Tapestry, JSF), NodeJS, PHP, ASP.NET, Rails, etc.

30

Independientemente del Lenguaje...



31

Lógica

- Funciones que operan sobre datos
- Generalmente stateless en memoria, statefull en la persistencia
- Dos enfoques:
 - centralizado (lightweight):
 - poco uso de CPU con operaciones "cortas"
 - objetos Java con métodos stateless (POJO)
 - fácil desarrollo
 - no requiere de servidores complejos
 - Ej: Spring, Hivemind, PicoContainer, etc.

33

Presentación

- **Fat client** (o Rich Internet Applications)
 - Procesamiento en el cliente (ordenamientos, filtrados, generación de reportes, ...)
 - Se apoyan en tecnologías adaptadas a la Web
 - Se basan en tecnologías de código móvil
 - Interfaces ricas similares a Desktop
 - Ej.: JS frameworks, GWT, Flash, OpenLazlo, Applets, ...
- **Thin client** (Web)
 - Poco procesamiento en el cliente (validación)
 - Interfaces pobres
 - Ej.: Servlets, JSP, Web Frameworks (Struts, Tapestry, Echo2, JSF), ASP.NET.

35

Persistencia

- Generalmente RDBMS
- Dos enfoques:
 - **Acceso directo a tablas/relaciones mediante SQL**
 - Gap semántico objetos/tablas
 - Dependencia RDBMS
 - Alto costo de desarrollo y mantenimiento
 - Ej: JDBC, EJB BMP, ADO.NET, ODBC
 - **Mapeo objeto/relacional (O/R)**
 - Objetos -> Tablas (generación de esquema)
 - Tablas -> Objetos (ingeniería reversa)
 - OQL (Object Query Language) en vez de SQL
 - Independencia RDBMS
 - Performance mejor si se usa bien (caching)
 - Hibernate, EJB CMP, JDO, Toplink, ...

32

Lógica (cont.)

- distribuido (heavyweight):
 - uso intensivo de CPU y operaciones "largas" (simulaciones, aplicaciones financieras)
 - componentes complejos
 - comunicación remota entre componentes
 - difícil desarrollo (necesidad de adherir a un modelo de componentes muy rico)
 - requiere de servidores complejos (transacciones distribuidas, balanceo de carga, redundancia, tolerancia a fallos, etc.)
 - Ej.: EJB, DCOM, CORBA.

34

Por qué es Diferente?

- Plataformas heterogéneas en los clientes (Firefox, Opera, Edge, etc.)
- Escalabilidad en número de usuarios
- Seguridad
- Tiempos de desarrollo cortos, muchos cambios por nuevas oportunidades de negocios
- Tecnologías inmaduras
- Separación diseñador/programadores?!?!?
- Tendencias: dispositivos móviles (PWA), automatización (servicios/semántica), operaciones desconectadas, ...

36

Requerimientos de las Aplicaciones Web

37

Availability

- Perceived uptime by a user
- 99% uptime represents 1% downtime is
 - 864 seconds/day or 14.4 minutes/day
 - 315,360 seconds/year or 5256 minutes/year or 88 hours/year

Downtime	Percentage Uptime
53 minutes/year or 0.14 minutes/day	99.99%
5 minutes/year	99.999%
30 seconds/year	99.9999%
3 seconds/year	99.99999%

39

Downtime Costs (per Hour)

• Credit card authorization	\$2,600,000
• Ebay (1 outage 22 hours)	\$225,000
• Amazon.com	\$180,000
• Package shipping services	\$150,000
• Home shopping channel	\$113,000
• Catalog sales center	\$90,000
• Airline reservation center	\$89,000
• Cellular service activation	\$41,000
• On-line network fees	\$25,000
• ATM service fees	\$14,000

Source: InternetWeek, 4/3/2000 • Fibre Channel: A Comprehensive Introduction, R. Xipell/2000, p.4. "Based on a survey done by Contingency Planning Research."

41

Requerimientos

- Availability
- Scalability
- Security
- Performance
- Integrity
- Manageability
- Malleability/Longevity
- Integration
- Cost

38

In the News

eBay
12 June 1999 outage: 22 hours
Operating System Failure
Cost: \$3 million to \$5 million revenue
net
26% decline in stock price

AT&T
13 April 1998 outage: Six to 26 hours
Software Upgrade
Cost: \$40 million in rebates
Forced to file SLAs with the FCC (frame relay)

America Online
6 August 1996 outage: 24 hours
Maintenance/Human Error
Cost: \$3 million in rebates
Investment: ???

Causes of Unplanned Application Downtime



Charles Schwab & Co.
24 February 1999 through 21 April 1999: Four outages of at least four hours
Upgrades/Operator Errors
Cost: ???; Announced that it had made \$70 million in new infrastructure investment. \$s

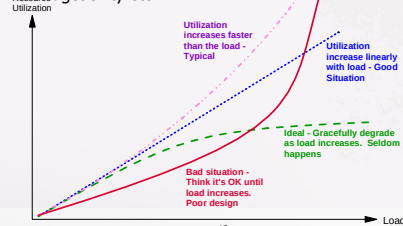
Dev. Bank of Singapore
1 July 1999 to August 1999: Processing Errors
Incorrect debiting of POS (due to a system overload)
Cost: Embarrassment/loss of integrity; interest charges

E*Trade
3 February 1999 through 3 March 1999: Four outages of at least five hours
System Upgrades
Cost: ???
22 percent stock price hit on 5 February 1999

40

Scalability

- The capability of a system to adapt readily to a greater or lesser intensity of use, volume, or demand while still meeting its business objectives:
 - acceptable levels of performance, availability, manageability etc.



42

Security

- Privacy
- Authentication
- Authorization
- Audit
- Non-repudiation

43

Integrity

- Data correctness
- Data permanence
- Disaster recovery
- Data currency

45

Malleability/Longevity

- Continuous availability (despite update and failure)
- Time period of use of program

47

Performance

- How long does it take to get a response to a request from the system?
 - Top-level metrics
 - Latency
 - Throughput
- How many transactions can be completed in a unit of time (Capacity)?
 - Subsidiary metrics
 - CPU
 - Network Bandwidth
 - I/O of various types, ...
- More performance:
 - Less hardware
 - More users, \$\$\$

44

Manageability

- Consider number of elements in a web applications
- Consistency
- Security
- Modifications
- Performance
- Configuration
- Training level required of operators

46

Integration

- Note: millions of person-years of spent every year for applications: multi-trillion \$\$\$
- Hence, integration is a necessity
- Integration approaches
 - Application to application
 - Data sharing by multiple applications
 - Process (Complex application integration)
- For some applications, integration cost is 7x cost of system, yet this is less than recreating existing applications or losing benefits of integrated systems

48

Cost

- Initial implementation
- Modification
- Installation
- Management (management is greater than development cost – usually at least double)

49

51

53

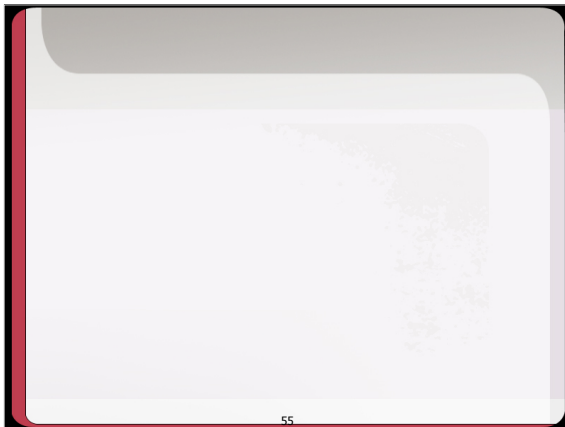
How do we Accomplish That?



50


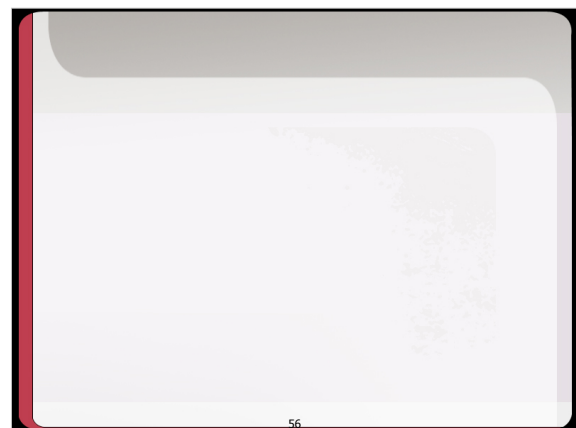
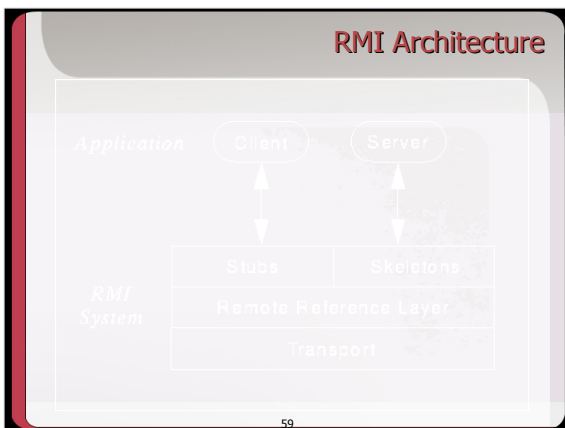
52

54



RMI - Remote Method Invocation

- It is the action of invoking a method of a remote interface on a remote object
- The method invocation on a remote object has the same syntax as the method invocation on a local object
 - local == distributed
 - `object.message();`
- Easier than sockets/HTTP/CGI...
- Transparent syntax:
- Very high level
- Object pass by value
 - JAVA serialization
- Heterogeneous environments

Design Goals

- Support seamless remote invocation on objects in different virtual machines
- Support callbacks from servers to applets
- Integrate the distributed object model into the Java language in a natural way while retaining most of the Java language's object semantics.
- No differences between the distributed object model and local Java object model
- Make writing reliable distributed applications as simple as possible
- Preserve the safety provided by the Java

Stubs and...

- Stub:
 - Initiating a call to the remote object (by calling the remote reference layer).
 - Marshaling arguments to a marshal stream (obtained from the remote reference layer).
 - Informing the remote reference layer that the call should be invoked.
 - Unmarshaling the return value or exception from a marshal stream.

...Skeletons

- Skeleton:

- Unmarshaling arguments from the marshal stream.
- Making the up-call to the actual remote object implementation.
- Marshaling the return value of the call or an exception (if one occurred) onto the marshal stream.

61

Using RMI

- the data type of any remote object that is passed as an argument or return value (either directly or embedded within a local object) must be declared as the remote interface type (for example, Hello) not the implementation class (HelloImpl)

63

An Example: Server

```
import java.rmi.Naming;
import java.rmi.RemoteException;
import java.rmi.RMISecurityManager;
import java.rmi.server.UnicastRemoteObject;

public class HelloImpl extends UnicastRemoteObject implements Hello {

    public HelloImpl() throws RemoteException {
        super();
    }

    public String sayHello() {
        return "Hello World!";
    }

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

65

Using RMI

- a remote object is an instance of a class that implements a Remote interface.
- A remote interface will declare each of the methods that you would like to call from other JVM
- RI have the following characteristics:
 - must be public. Otherwise, a client will get an error
 - extends the java.rmi.Remote interface
 - each method must declare java.rmi.RemoteException

An Example: Server

```
package examples.hello;

import java.rmi.Remote;
import java.rmi.RemoteException;

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

64

An Example: Server

```
try {
    HelloImpl obj = new HelloImpl();

    // Bind this object instance to the name "HelloServer"
    Naming.rebind("//myhost/HelloServer", obj);

    System.out.println("HelloServer bound in registry");
} catch (Exception e) {
    System.out.println("HelloImpl err: " + e.getMessage());
    e.printStackTrace();
}
}
```

66

An Example: Client

```
import java.applet.Applet;
import java.awt.Graphics;
import java.rmi.Naming;
import java.rmi.RemoteException;
public class HelloApplet extends Applet {
    String message = "blank";
    Hello obj = null;
    public void init() {
        try {
            obj = (Hello)Naming.lookup("//" +
                getCodeBase().getHost() + "/HelloServer");
            message = obj.sayHello();
        } catch (Exception e) {
            System.out.println("HelloApplet exception: " + e.getMessage());
            e.printStackTrace();
        }
    }
    public void paint(Graphics g) {
        g.drawString(message, 25, 50);
    }
}
```

67

Parameter Passing

- An argument to, or a return value from, a remote object can be any Java object that is serializable.
 - Java primitive types
 - remote Java objects
 - by reference. A stub is passed
 - non-remote Java objects that implement the `java.io.Serializable` interface
 - serialized and passed by copy
 - Classes, for parameters or return values, that are not available locally are downloaded dynamically by the client.

Compiling and Running

- Skeletons/Stubs:
 - `rmic HelloImpl`
 - `HelloImpl_Stub.class`
 - `HelloImpl_Skel.class`
- Starting the RMI Registry
 - `rmiregistry` &
- Starting the server
 - `java HelloImpl`
 - The output should look like this: `HelloServer` bound in registry
- `appletviewer hello.html` &

68