Module-IV

Server-side programming in Java, Pervasive web application architecture, Device independent web application.

Server-side programming in Java:

Client Side vs. Server Side Web

- Simply defined, client-side code executes on the end-user's computer, usually within a web browser.

  Server-side code executes on the web server, usually within a web application environment, which in turn generates HTML to be viewed in a browser.

- In general, the key areas where client-side coding has advantages stem from its location on the user desktop and/or other end device. They include the following:
  - Interactivity (e.g., mouse and keyboard handling)
  - Handling of user interface controls: buttons, textboxes, etc.
  - Feedback and validation

  Key server-side strengths include stem from their proximity to the backend business databases and other applications. They include the following:

  - Direct information access, retrieval, processing and storage , facilitate e-commerce, reservations, shipment tracking etc.
  - central repository of added web features such as e-mail, chat and multimedia streaming
  - security and authentication (mostly)

<u>Client Side Technologies</u>

- HTML : markup language for display of web content
  - DHTML extensions for dynamic and interactive control of web page content and display (Not fully standardized by W3C yet)
  - Tools for writing html documents include : Dreamveawer, FrontPage and any word processor (including Notepad)
- JavaScript: client side programming language
- VBScript: client side programming language (MS proprietary, supported by IE)
- Java Applets:
  - small programs written in Java, embedded in an HTML page and executed from within a browser
  - Unlike JavaScript, the Java code must be pre-compiled into a so-called bytecode before it can be interpreted by a browser's so–called Java Virtual Machine
  - In other words, the Notepad and the browser alone are not enough to write java applets

- ActiveX controls
  - Similar to Java Applets but can be written in a variety of programming languages such as C, C++, VB and even Java
  - Supported by Windows only
  - Security issues: unlike Java applets, ActiveX controls have full access to all desktop resources: memory, operating systems, …,Authentication and registration system

- Macromedia Flash

    - Proprietary commercial application for creating interactive graphic content
    - It has its own scripting language
    - To reproduce the Flash content browsers must be equipped with a Flash Player plug-in

- Client Side technologies have evolved form a simple tools for creating static pages to sophisticated array of technologies turning a browser into a powerful multifunctional client.
- Consequently, we can stop referring to a web client as "thin" client (i.e. limited in size and computational needs**).**

Server Side Technologies

Server-side technologies are quite numerous and diverse. Popular server side web application technologies include:

- Microsoft ASP/.NET
- Java server technologies such as J2EE, JSP, and servlets
- CGI / Perl
- PHP
- ColdFusion
- In addition, the server-side technologies include database systems such as Oracle, SQL Server (Microsoft), MySQL (open source) and many others
- .NET is Microsoft framework supports many programming languages such as VB, C++, C#, JScript
- ASP.NET (Active Server Pages) is an integral part of .NET initiative
    - It is a technology for creating dynamic web content on the server that appears as HTML on a client's browser
    - Developers can use this technology to write scripts in a language of their choice for from processing, interactive web pages, or any other dynamic content
    - Every element in an ASP.NET page is treated as an object and run on the server.

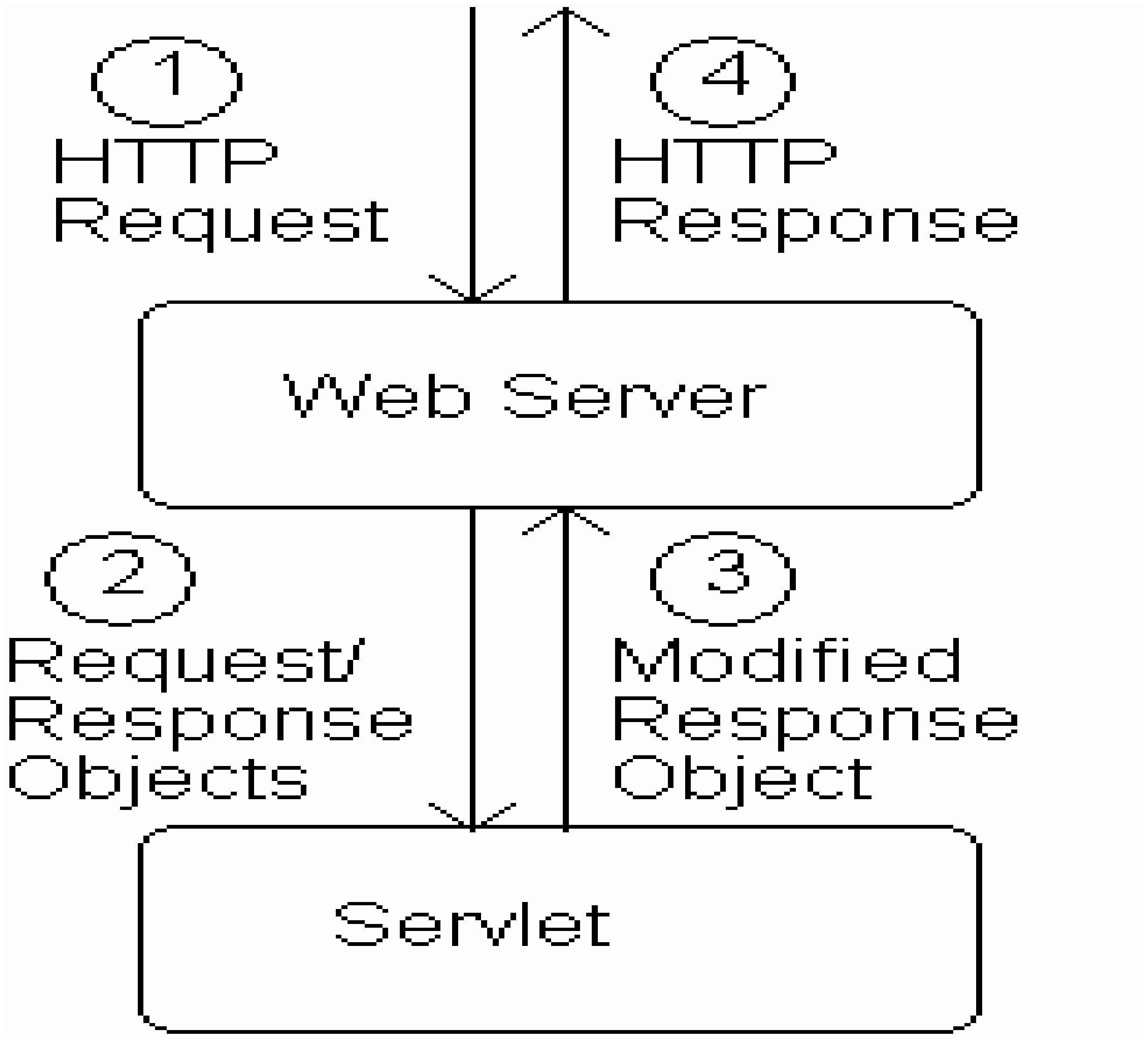
- ASP.NET server controls are components that can perform the same work as HTML controls: radio buttons, text boxes, buttons, etc.

Unlike HTML controls, ASP.NET controls preserve their content if and when this is needed

Server-side Programming: Java Servlets

- Similarly, web server response can be static or dynamic
  - Static: HTML document is retrieved from the file system and returned to the client
  - Dynamic: HTML document is generated by a program in response to an HTTP request
- Java servlets are one technology for producing dynamic server responses
  - Servlet is a Java class instantiated by the server to produce a dynamic response

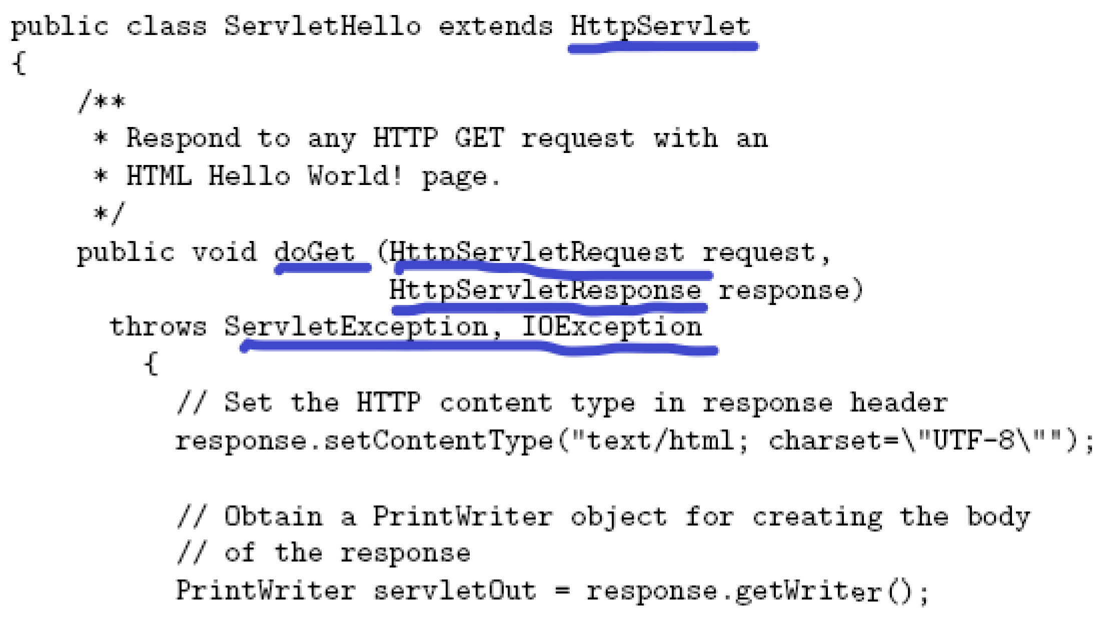
Servlet Overview:



When server starts, it instantiates servlets

1. Server receives HTTP request, determines need for dynamic response
2. Server selects the appropriate servlet to generate the response, creates request/response objects, and passes them to a method on the servlet instance
3. Servlet adds information to response object via method calls
4. Server generates HTTP response based on information stored in response object

Example:- Hello World! In Servlet



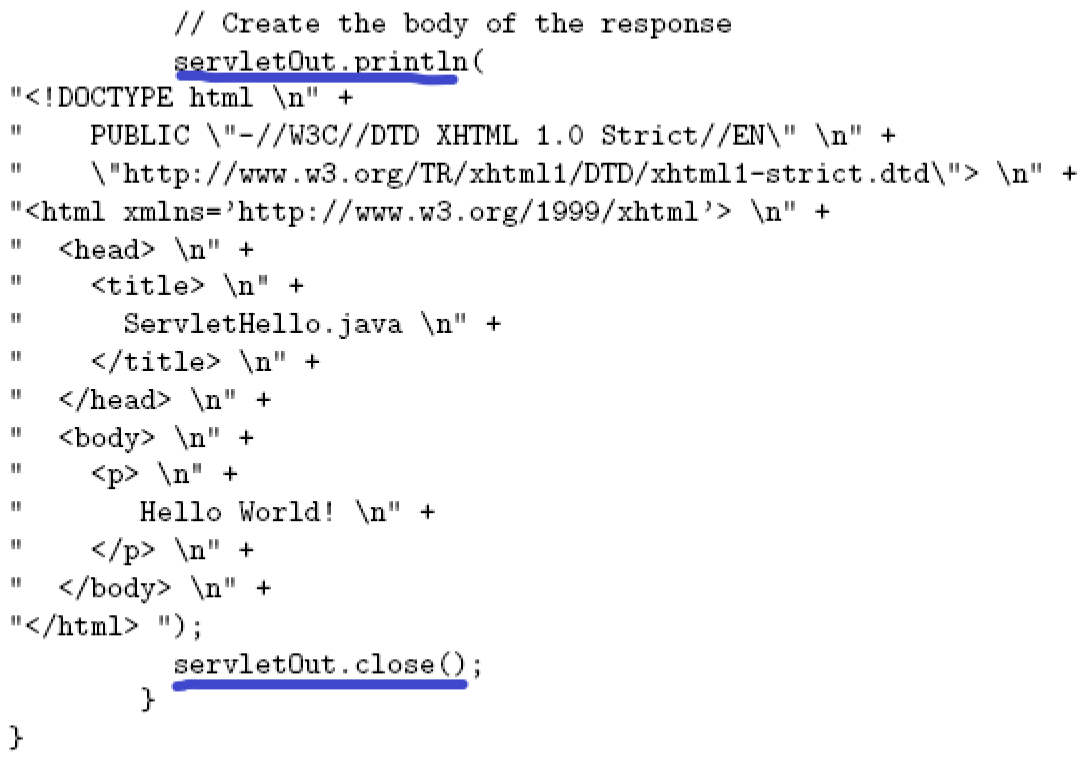
```
public class ServletHello extends HttpServlet
{
    /**
     * Respond to any HTTP GET request with an
     * HTML Hello World! page.
     */
    public void doGet (HttpServletRequest request,
                       HttpServletResponse response)
      throws ServletException, IOException
      {
        // Set the HTTP content type in response header
        response.setContentType("text/html; charset=\"UTF-8\"");

        // Obtain a PrintWriter object for creating the body
        // of the response
        PrintWriter servletOut = response.getWriter();
```

HttpServlet→ All servlets we will write are subclasses of HttpServlet.

doGet→ Server calls doGet() in response to GET request

HttpServletRequest / HttpServletResponse→ Interfaces implemented by request/response objects.

ServletException / IOException→ Production servlet should catch these exceptions



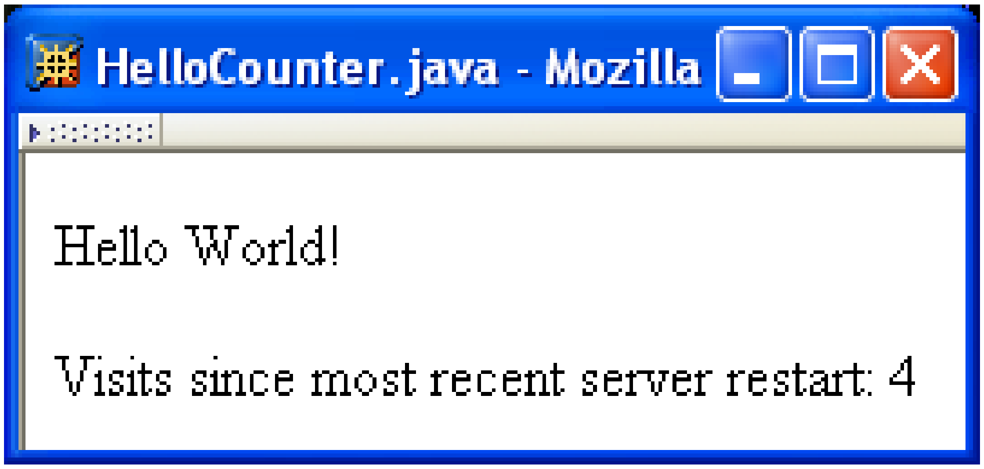
```
          // Create the body of the response
          servletOut.println(
"<!DOCTYPE html \n" +
"    PUBLIC \"-//W3C//DTD XHTML 1.0 Strict//EN\" \n" +
"    \"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd\"> \n" +
"<html xmlns='http://www.w3.org/1999/xhtml'> \n" +
"  <head> \n" +
"    <title> \n" +
"      ServletHello.java \n" +
"    </title> \n" +
"  </head> \n" +
"  <body> \n" +
"    <p> \n" +
"       Hello World! \n" +
"    </p> \n" +
"  </body> \n" +
"</html> ");
          servletOut.close();
        }
}
```

servletOut .println(→ HTML generated by calling print() or println() on the servlet's PrintWriter object

servletOut.close() → Good practice to explicitly close the PrintWriter when done

Running Servlets

- Simple way to run a servlet :
    1. Compile servlet (make sure that JWSDP libraries are on path)
    2. Copy .class file to shared/classes directory
    3. (Re)start the Tomcat web server
    4. If the class is named ServletHello, browse to http://localhost:8080/servlet/ServletHello



Hello World!

Visits since most recent server restart: 4

- Servlets do not have a main()
  - The main() is in the server
  - Entry point to servlet code is via call to a method (doGet() in the example)
- Servlet interaction with end user is indirect via request/response object APIs
  - Actual HTTP request/response processing is handled by the server
- Primary servlet output is typically HTML

Pervasive web application architecture:

Pervasive Computing:

Pervasive Computing is computing power that enables software applications available anytime and anywhere. Personal Digital Assistant(PDA) and cell phones are the first widely available and used pervasive computing devices.

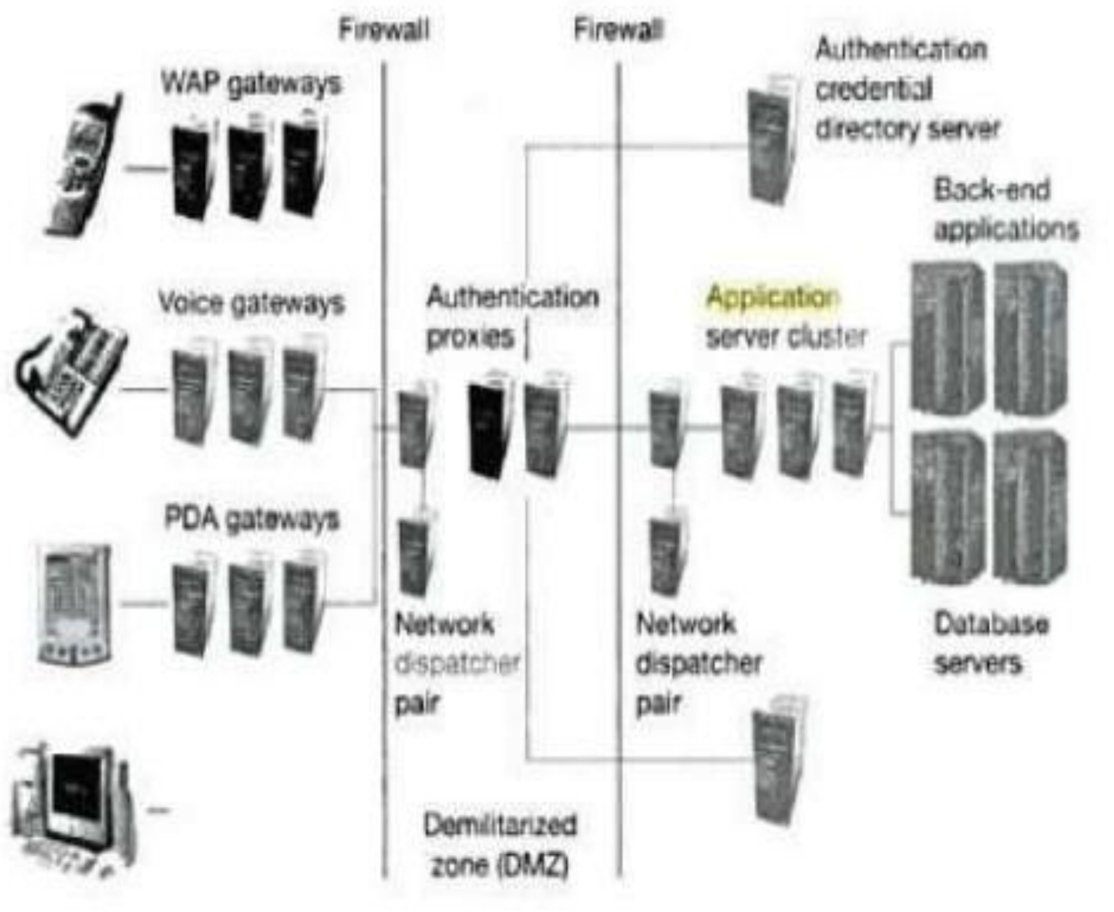
The architecture for pervasive computing applications that support multiple devices, such as PCs, WAP phones, PDA and voice-recognition devices.

The architecture addresses the special problems associated with pervasive computing, including diversity of devices, markup language and authentication methods shows how pervasive computing applications based on this architecture can be secured.

Examples of several kinds of pervasive computing devices include WAP phones, PDAs, and voice-recognition devices. These devices proving different user interfaces, use different markup languages, use different communication protocols, and have different ways of authenticating themselves to servers.

Scalability and Availability are the two important issues because pervasive architecture should as scalable to meet the large amount of user who are subscribing for applications. According to availability, when a user accessing the application and if it is not available then the user will assume that it does not works and the user will switch on to next service. Thus the service should be available whenever needed

A scalable topology for pervasive computing is shown in the Fig. This shows several gateways used for accessing the server.



A Scalable Topology for pervasive computing web applications

- WAP gateway which executes WTLS protocol in the direction of clients and SSL in the direction of servers.
- Voice gateways use voice recognition engine which consumes more power.
- PDA gateway for the PDAs.
- Network Dispatcher which routes incoming request to the appropriate server. Support handling of HTTP request from a particular client is always sent to the same server to avoid repeated SSL handshakes.
- Two dispatchers available to increase availability.
- Two firewalls are placed to perform secure connection.
- Authentication proxy is placed which checks all incoming request according to security policy defined and use the credentials given by the client in order to secure transactions. Authentication proxy consumes significant computing power.

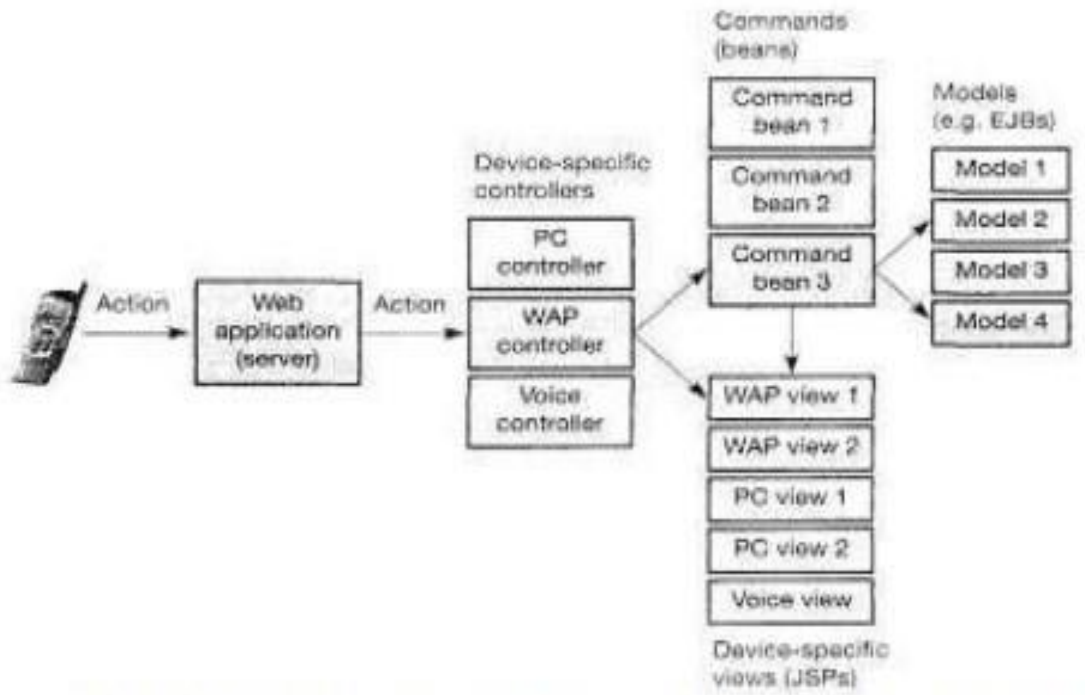
- Cluster of application servers are arranged in order to add additional machines if load increases.

Business Logic Designers, User Interface Designers, Application Programmers and experts for existing database system are the roles assigned to implement web application**.** Application flow is designed by Business Logic Designers. Look and feel of the system is given by User Interface Designers. Programmers are concerned with technologies such as HTML and JSP and implementing the application logic. Those experts who monitor the gateways are responsible in knowing the technologies such as WML, Voice XML.

Pervasive Application Architecture:

The model-view-controller (MVC) pattern is a good choice when implementing Web applications.

Standard mapping of the pattern to Java servlets, JSPs(Java Server Pages), and EJBs(Enterprise Java Bean*)*, where controller is implemented as a servlet, the model implemented as a secure EJBs, and the views as JSPs.



MVC Pattern applied to Pervasive Computing Applications

As devices are very different from each other, we can assume that one controller will fit all device classes. In the MVC pattern the controller encapsulates the dialog flow of an application.

This flow will be different for different classes of devices, such as WAP phone, voice-only phones, PCs, or PDAs.

Thus, we need different controller for different classes of devices.

To support multiple controllers, we replace the servlet's role to that of a simple dispatcher that invokes the appropriate controller depending on the type of device being used.

Device independent web application:

Device independent components work right no matter what model of device you use them with. For example, if the graphic file format in your publication is device independent, the results you see on paper will look about the same whether you print to an HP DeskJet, an Apple LaserWriter, or a high-resolution Linotronic image setter (the graphic will be printed at whatever resolution the printer uses).

Device dependent components by contrast, work right only with a particular model of the device. A device dependent bitmap graphic looks the way it's supposed to only on a particular type of monitor-on other screens it looks funny or may not display at all.

All the programs that run under Windows and the Macintosh are device independent. A given program doesn't have to worry about how to work with every mouse, keyboard, printer, screen, or scanner on the market. Instead, as long as the environment has a software control module (or driver) for the devices in your system, every program will automatically produce the expected results. Of course, there may be some variations due to the differing capabilities of different models. For instance, text you print on a 300-dot-per-inch laser printer will not be as smooth as text from a 2540-dot-per-inch image setter. But the basic look of the text will still be the same if you have a device independent system.

The Web is steadily increasing its reach beyond the desktop to devices ranging from mobile phones to domestic appliances. This rapidly expanding accessibility is largely due to the Web's foundation in open protocols and markup languages, which offer the most widely implemented global infrastructure for content and application access.

HTML's original aim was to provide a device-independent markup language that was based on document semantics. It identified document elements such as headings, paragraphs, and lists without specifying presentation.

Early on, however, browser developers introduced many ad hoc presentation-specific elements and attributes to HTML that blurred the distinction between semantics and presentation.

A presentation created for a large-screen device, for example, can theoretically be displayed and interacted with on a small-screen device because they use the same markup.

Practically, however, it might be too hard to use simply because the author did not create the presentation with a small form-factor in mind.

Web standards are now encouraging a renewed distinction between semantics and presentation through styling languages such as Cascading Style Sheets (CSS) and Extensible Stylesheet Language Formatting Objects (XSL-FO) for addinginformation to Web output, and interaction markup languages such asXML Forms (XForms) for input.

Ideally, authors would need to create only one version of their Web content. Then, during the delivery and rendering process, adaptation software would create a presentation to match the delivery device's capabilities.

Here we typically design Web applications for the PC browser and screen. To exert maximum control over the final appearance, they often base Web page layouts on tables, specified using absolute pixel positioning. Because adapting such a site for a small display is effectively impossible, we must create a parallel site to accommodate these devices.

Figure 1 shows similar content adapted for different devices. As the variety of Web-connected devices increases, however, creating a separate site for each kind of device is both economically and administratively impractical.

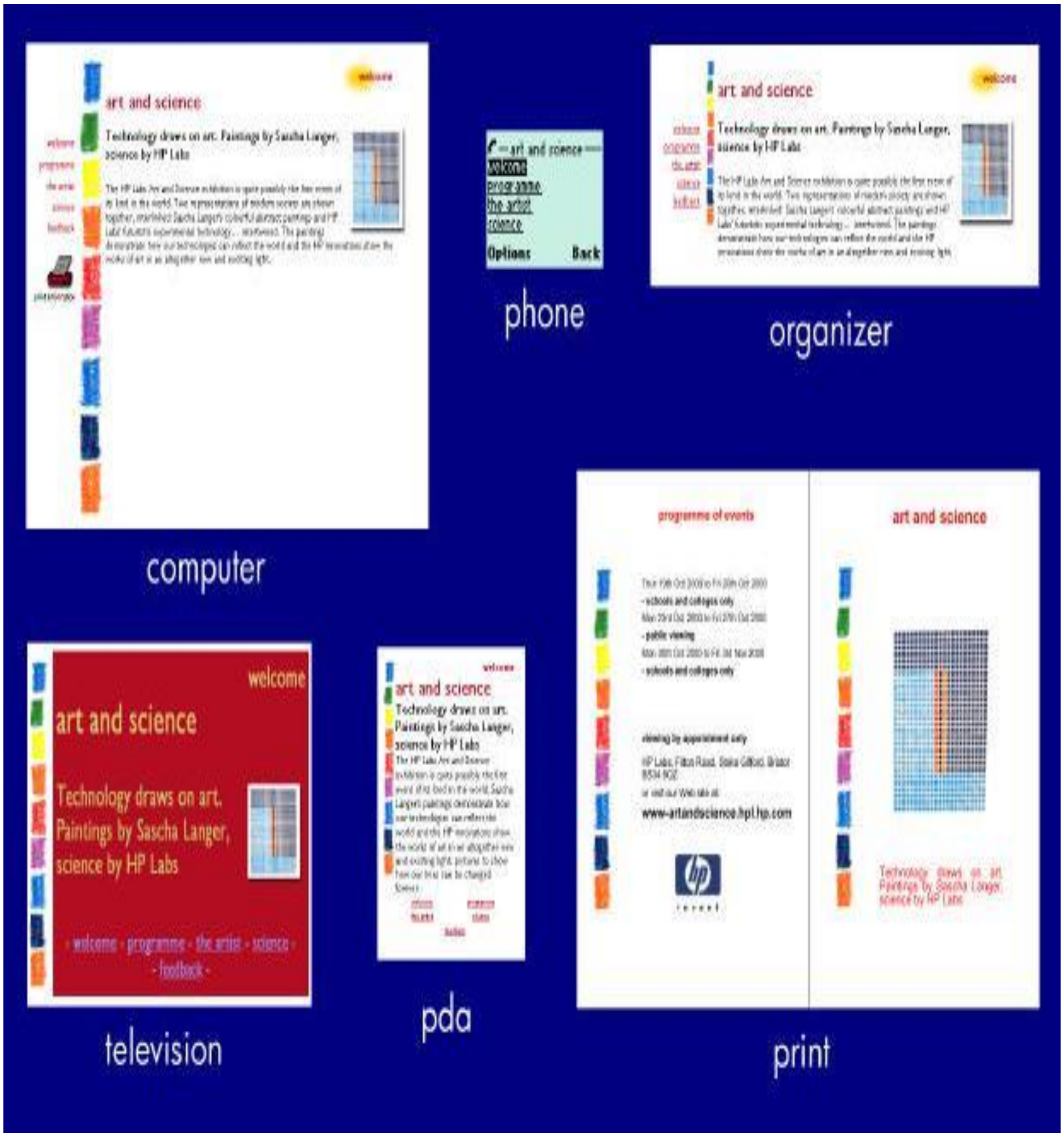


Figure 1: Content adapted for different devices. Although device display capabilities overlap in some cases, other factors, such as text entry capabilities, can vary widely.

Currently, designers are using many different approaches to achieve better device independence for Web presentations. The existing approaches fall into three broad categories — intermediate, client-side, and server-side — depending on who controls the adaptation process.

Intermediate adaptation

To avoid changing either the server that provides content or the client that consumes it, intermediaries in the content delivery chain can offer limited adaptation. Transcoding proxies, for example, can transform image formats or even subsets of markup languages. This gives data-enabled phones access to Web sites by either omitting a server's full-resolution color images or transcoding them into low-resolution or monochrome versions, depending on the phone's display capabilities.

Client-side adaptation

Adaptation can also occur in the content delivery device (typically the Web browser). The advantage here is that the adaptation code usually has direct access to the device's capabilities.

Some client-side adaptations are independent of content directives. Many browsers, for example, let users increase or decrease text display size.

Server-side adaptation

Server-side adaptation offers maximum control over the delivered content, including the ability to radically change the content amount and its styling, navigation, and layout. In order to produce the most appropriate adaptation, however, the server must have sufficient information about the delivery context, including the delivery device's capabilities.

There are many techniques for achieving server-side adaptation. Web site designers have been delivering different versions of their content for years to accommodate nonstandard browser implementations.