Advanced Internet Programming

Architecture

Chris Wong
chw@it.uts.edu.au
Topics

• Generic Architectures
  – Client/Server
  – Three-tier
  – N-tier
  – Sample configurations

• Technology Architectures
  – Java EE
  – Microsoft .NET
Motivation

- Architectures:
  - abstract away from implementation details
  - focus on design principles
  - allow comparison of seemingly different applications
  - see the “big picture”
Tiered architectures

• One way of classifying enterprise systems is based on the number of tiers

• In principle, each “tier” represents a different system executing one piece of the overall application

  – **One-tier**: non-networked application
  – **Two-tier**: client/server application
  – **Three-tier**: business logic is distinct tier
  – **N-tier**: distributed application
One Tier

• Advantages:
  – simple

• Disadvantages:
  – no resource sharing
  – monolithic applications
    • needs powerful computer
    • difficult to maintain
    • difficult to integrate
  – not scalable
  – etc.
Two Tier

Client

Server
Two Tier

• Examples:
  – web browser (client) + web server (server) viewing static HTML pages
  – custom client + database server applications (e.g. order processing, inventory, etc)
Two Tier

**Advantages:**
- shared access to data, resources
- typically robust, reliable (over private network)
- less complex

**Disadvantages:**
- multiple requests for data - tax the network
- difficult to maintain or extend functionality
- upgrade of custom client is labour-intensive
- exposes to security violations
- not scalable (especially to web audiences)
- ties to one presentation type
Three Tier

User Interface

Client PC

Business Logic

Application Server

Data

Database Server(s)
Other Data Source(s)
Three Tier

• Examples:
  – web browser (user interface), CGI program on web server (business logic), database (data)
  – Java Swing client, network connection to remote web services server (business logic) which accesses a database
Three Tier

- **Advantages:**
  - separation of business logic as distinct tier makes maintenance easier
  - multiple user interfaces can be built and deployed
  - supports applications that use multiple data sources
    - enterprise database, XML documents, directory service
  - encourages applications to reuse data sources

- **Disadvantages:**
  - complexity
  - vendor incompatibility
  - single point of failure
N-Tier

• Examples:
  – component-based application implemented using a web application server
  – e.g. Java Enterprise

  – the focus in this subject
N-Tier

• Advantages:
  – supports distributed applications
  – applications are built from reusable *components*
  – highly scalable

• Disadvantages:
  – complexity
  – reuse?
  – cost
  – performance
  – load balancing issues - complex
Which architecture?

- Depends on size of application:
  - One-tier: cannot share data/resources
  - Two-tier: only 1 data source, simple website
  - Three-tier: small application, future integration & reuse of business logic unlikely
  - N-tier: most flexible solution, but complex (more hardware, software, skills)
B2C: Simple 2-tier café site

1. User clicks on menu link (on Index.html) to see dates for upcoming events at the Cafe. (Date.html)

2. HTTPRequest for Date.html

3. HTTPResponse with Date.html

4. Date.html displayed in browser
B2C: 3-tier chat room site

1. User registers on chat room by completing registration form and then clicks on "Submit" button.

2. HTTPRequest for form.jsp

3. SQL command from form.jsp to save details to the database

4. Data saved to database

5. SQL Result ie saved

6. HTML (maybe with Javascript) + JSP

7. form.jsp displayed in browser

Client(s)

Web or Web Application Server

Browser on Workstation

Data Layer

Business & Presentation Layer

Form.jsp displayed in browser
B2C: n-tier grocery shopping site

1. User clicks on menu link to "checkout"

2. HTTPRequest for checkout.jsp with cart details

3. checkout.jsp runs and sends on cart details

4. Cart details (typically sent with RMI)

5. Java saves details to database

6. SQL query with quantity of products ordered.

7. Data saved to database

8. Reply from database

9. Checkout details calculated and sent on.

10. Check out details

11. Data Layer

12. checkout.jsp displayed in browser
n-tier can get complex...!

CLIENT SERVICES

WEB "CONTACT" SERVICES

APPLICATION SERVICES

CORPORATE SERVICES

DATA SERVICES

BROWSER

PDA

INTELLIGENT APPLIANCES

INTERNET SERVICE PROVIDER

LOAD BALANCER

WEB SERVER

FIREWALL

WEB SERVER

Content Management systems, Workflow

Application Servers, ORB, TPM

transaction server

LEGACY SYSTEMS

ERP

DATA BASE

DATA BASE

Security, Systems Management

© UTS 2009 – Architecture

Architecture-19
Components/Services

Infrastructure

- Operating System
- Web Server
- Application Server
- Database Server
- Directory Server
- Load Balancer
- Firewalls
- Proxies/caches

Application

- Search Engines
- Content Management
- Commerce servers
- Personalisation and Profiling servers
- Scenario servers
- Traffic Analysis servers
- Site Management servers
- Monitoring servers
- Streaming multimedia
Technology architectures

- Vendors provide different development and deployment environments
- Programmers want to code in different languages
- Programmers need support for building complex applications
  - database interaction
  - transactional interactions (CICS, TUXEDO)
  - application and data security
  - etc.
Before standard architectures

• Vendors would provide a **proprietary** software product (development/deployment environment)

• Programmers would write their code to suit the vendor’s product
  – *Locked* into that one vendor’s product
  – *Difficult to integrate* applications written using different vendor products
The current situation

Standards

- There are two main de-facto standard architectures for developing enterprise applications

1. Java Enterprise (aka Java EE or Java EE) (& variations e.g. Spring)
2. Microsoft .NET

- Of course there are other alternatives eg:
  - Cold Fusion, PHP, etc
Java EE Overview

• Java EE is a specification, not a product
  – it is implemented by a collection of products
  – specifies roles and interfaces to build applications
  – a runtime infrastructure for managing applications

• The Java EE specification describes:
  – A component model
  – Container requirements
    • Containers hold and manage components.
  – Standard services and APIs
    • Database, Transactions, Security,... etc
Java EE Components & Containers
Java EE – Components

• A component is an application-level software unit
  – Like JavaBeans in Java SE

• Java EE platform supports the following components:
  *Client based:*
  – Applets eg: Browser based
  – Application clients eg: Swing or AWT Java applications,
  *Server based:*
  – Web components eg: static or dynamic web pages
  – Enterprise JavaBeans eg: encapsulated logic
Java EE – Containers

- Components need an execution environment
  - This is called a **Container**

- A container is an application that runs, manages the lifecycle and provides services to components
  - The **Web browser** is a container for Applets
  - The **Java runtime** + operating system is the container for Java applications
  - The **Web server** is the container for Servlets/JSPs
  - The **Application server** is the container for EJBs
Java EE – Containers

- For convenience, many vendors have combined these into one package:

  *The Web Application Server*
Java EE – Containers

• Containers also provide some support services:
  – database connection pooling
  – container managed persistence
  – local naming and directory services
  – integration with transaction services
  – integration with messaging services
• Java EE platform consists of 4 containers
  1. Applet container - provides environment for Java applets
  2. Application-Client container - provides environment for Java EE application clients
  3. Web container - management of servlets and JavaServer pages
  4. EJB container - management of EJB
Java EE – Container Architecture

- A container can be divided into 4 parts:
  1. component contract
  2. container service API
  3. declarative services
  4. other container services
Java EE – Container Architecture

• 1. component contract
  – applications should be developed according to a defined framework
  – Applet is a good example
    • applet container calls init(), start() … and other methods
    • you extend Applet class - that allows it
  – Servlet is another example

• 2. container service API
  – allows access to various services, eg., JNDI, JMS
  – provides a consolidated view of various APIs
Java EE – Container Architecture

• 3. declarative services
  – deployment descriptor defines the contract between the container and component
  – explicit invocation vs declarative invocation
  – a declarative service is a service performed by the container for us

• 4. other container services
  – lifecycle management
  – resource pooling
  – populating the namespace
  – clustering
Java EE Architecture

• Let’s fit the Java EE component/container model together with the n-tier architectures....

• The Java EE architecture tends to favour the Gartner “Distributed Presentation” model.
  – Shares presentation logic between the browser and a web “front-end” server tier.
    • Note: client application can call business logic directly using RMI, Corba or web services
  – Places business logic in separate tier
  – Data access/Enterprise Information systems in yet another tier
Java EE & n-Tier model
Java EE Services

• Java EE provides many services for components to use. This includes:
  – JMS – Java Message Service
  – JNDI – Java Naming & Directory Interface
  – JTA – Java Transaction API
  – JavaMail & JAF - Java Activation Framework
  – RMI-IIOP – Java Remote Method Invocation, CORBA IIOP
  – JDBC – Java Database
  – JCA – Java Connectors
  – JAX-* – Java XML & Web services
Java EE – Servlets & JSP

• Servlets/JSP are responsible for “presentation logic”

• Creation and delivery of the user interface to the client
  – servlets are similar in invocation to CGI
  – JSP is an embedded language, like PHP or ASP

• Servlets and JSP consist of Java code that executes as part of a web server process and generates HTML/CSS/JavaScript that is sent to a client via HTTP
Java EE – Databases

• **JDBC** – Java Data Base Connectivity

• For writing **database-enabled** applications

• JDBC is an API that allows access to databases from within Java code
  – The API is generic and works with a variety of database implementations
  – Typically access databases from EJBs, although can be done from servlets/JSP (or even from applets, though usually a poor architectural choice)
Java EE – Directories

• **JNDI** – Java Naming and Directory Interface

• For **accessing directory** servers, particularly LDAP
  – LDAP = Lightweight Directory Access Protocol

• JNDI is an API that allows access to LDAP directories from within Java code
  – the concept is similar to using JDBC for accessing databases, although obviously the details are different
Java EE - Transactions

- **JTA** – Java Transaction API – allows Java applications to participate in **ACID transactions**
  - (ACID = Atomicity, Consistency, Isolation, Durability)
- Allows applications to declare unit’s of work
  - Providing Commit/Rollback calls
- Declarative or Programmatic
- Often Container can use legacy transaction managers
Java EE - JMS

- Use the **JMS (Java Message Service)** for asynchronous programming model and legacy Message Oriented Middleware integration eg: IBM MQseries, MSMQ
- Supports
  - point-to-point (queuing) programming model
  - publish/subscribe (pub/sub) programming model
- Not related to Mail or instant-messaging (ICQ etc)!!
Java EE - Mail

- **JavaMail** is a standard API to invoke email services – both send and receive mail

- Additional API – **JAF - Java Activation Framework** – allows JavaMail to also deal with MIME attachments
  - Also allows “multipart messages” – eg: can have Images, Binary, text files attached to mail

  - We don’t cover this in this course... 😞!
Java EE – Connectors

- Java EE Connectors are used for integrating Java applications with existing systems

- Java EE defines a Connector Architecture to provide a standard mechanism for integration

- There are also other options for integration that we will explore, e.g. CORBA, messaging services
Developing Java EE Applications

1. Develop application components
   – ie: model business rules in the form of components

2. Compose components into modules.

3. Compose modules into applications
   – ie: integrate multiple modules

4. Deploy & Install
Components into Modules

- **Web Modules**
  - consists of Java Servlets, JSP, HTML/XML etc.
  - packaged is called Web Archive File (**WAR**)  
  - `web.xml` is the deployment descriptor

- **EJB Modules**
  - consists of EJBs, JARs and resources into **JAR** files 
  - `ejb-jar.xml` is the deployment descriptor

- **Java Modules**
  - Java client classes into **JAR** files 
  - `application-client.xml` is the deployment descriptor
Modules into Applications

• **Application**
  - Consists of one or more WAR & JAR files
  - Packaged into an Enterprise Archive (EAR) file
  - `application.xml` is the deployment descriptor
  - Is the highest level of Java EE packaging
  - Is an independent unit of code

• Java EE applications can call and invoke each other too...
Application Deployment

• Prepare the application for installing
  – depends on each vendor’s implementation
  – Usually by copying the EAR file to a run-time directory
  – Most vendors supply a deployment tool
  – Vendor may generate additional supporting classes

• Configure
  – creating data sources & connection pools
  – creating connection factories
  – various tuning and runtime parameters
Application Development Roles

• Java EE platform implementer
  – vendor usually does this – or open source!
  – Sun provide reference platform for free

• Application Component Provider
  – application developer

• Application Assembler
  – package application components

• Deployer
  – deploy packaged applications and configures
Application Development Roles (2)

• Tester
  – performs unit and integration testing

• System Administrator
  – administers the application

• Tool Provider
  – provides tools for the development
Benefits of Java EE Architecture

- To the Application Developer (you)
  - simplicity
  - portability
  - reusability
  - simplifies building of complex applications
  - separation of business logic from presentation logic
  - deployment in many operating environments
  - distributed deployment
  - integration with non-Java systems
  - educational resources
Benefits of Java EE Architecture

• To Customers
  – choice of server
  – application management tools
  – integration with existing resources
  – security
Microsoft .NET

- Java EE is developed by Sun Microsystems under an open process ("Java Specification Request")
- Main competitor to Java EE architecture is Microsoft .NET

Microsoft .NET

- Supports roughly equivalent concepts to Java EE
.NET Overview

• .NET is a set of specifications:
  – Implemented as collection of products
  – Some of the services required are part of base operating system (Windows 2000/2003 server, Windows XP)

• The .NET specification includes:
  – component model
  – container requirements
  – standard services and APIs
    (Database, Transactions, Security,...)

→ Sound familiar?
.NET ++

• .NET is also a set of Microsoft products designed to work smoothly together.

• UNLIKE Java EE, includes:
  – Smart client software – eg: .NET framework on PC’s, mobile devices (.NET Compact Framework), embedded (.NET Embedded)
  – XML services – tightly integrated XML & web services
  – Enterprise services – eg: Biztalk server
  – Systems Management
  – Development toolset – Visual Studio.NET

➔ You could buy 3rd party tools for Java EE though… 😊
.NET Architecture

Client Browser (Internet Explorer, Netscape, etc)

Web Server (IIS running under Windows)

Presentation Layer

ASP.NET Web Forms (HTML, custom tags, client-side script)
Responsible for generating HTML and accepting input

ASP.NET Code-Behind Classes (C#, VB.NET, J#, managed C++, etc)
Responsible for handling user events and calling the business layer

Application Server

.NET Managed Classes (C#, VB.NET, J#, managed C++, etc)
Responsible for implementing business logic and data access logic for business entities

ADO.NET
Responsible for abstracting the database vendor and providing connection pooling

Business Layer

Data Layer

Database Server (Windows, UNIX, LINUX, etc)

Database Server (SQL Server, Oracle, etc)

Stored Procedures
Data access is typically provided through stored procedures to allow for improved security and performance.


© UTS 2009 – Architecture
.NET architecture

- Like Java EE, Microsoft .NET consists places components and containers in n-tier layers.

  - **Presentation Layer**
    - using ASP.NET web forms & embedded code (.aspx files)

  - **Business Logic layer**
    - Using Managed classes (ActiveX objects & DLL’s)

  - **Data access layer**
    - Using ADO.net (and ODBC)
.NET Run-Time

- .NET requires a run-time framework installed on the host OS
- Supports multiple langs via CLR and MSIL
- Common class libraries shared by all languages
.NET Framework

- Microsoft .NET supports multiple languages by compiling these languages into an Intermediate Language (IL or MSIL).
- This MSIL is interpreted by the Common Language Runtime (CLR).
- .NET framework also provides a common framework of classes which all supported languages can use (e.g., C#, VB.net, J#, managed C++, etc).
- .NET also defines a Common Type System (CTS) for common object data types.
- Restricted to Windows environment (but open source .NET framework, mono, under development).

→ Much like Java with its Runtime Environment (JRE) and standard Java API’s
• **ASP** – Active Server Pages (on IIS Web Server) are responsible for “presentation logic”

• Creation and delivery of the user interface to the client
  – ASP and JSP have many similarities

• ASP consists of scripting code (C#, VBScript, JScript) & custom tags embedded with HTML

• ASP dynamically generates appropriate HTML/CSS/JavaScript pages.
.NET – Components

- Business logic in .NET is provided by components
- Server-side components are implemented as managed objects
- Can re-use COM/DCOM objects
- .NET runtime supports transactions – manual and automatic
- .NET remoting allows objects to be shared between applications and servers.

→ Unlike Java EE, objects are not distributed automatically. Scale by adding more web servers... 😞
.NET – Databases

- Developers use ADO.NET to connect to databases
- This can be SQL based using ODBC (Open Database Connectivity) or XML

- ADO – Active Data Objects. This provides direct object ↔ database mapping.

- ODBC is an API that allows access to databases from within Microsoft (C#, VB, etc.) code
  - The API is generic and works with a variety of database implementations
.NET – Other services

• The .NET architecture describes a number of servers which are used:
  – Application Center (for deployment management)
  – SQL Server (for database)
  – Exchange Server (for email/messaging)
  – Commerce Server (for e-business functionality)
  – Transaction Server (for transactional components)
Java EE vs .NET

• They are two different architectures for accomplishing the same task

• One is based around Java
  “One language, Many platforms”

• One is based around Microsoft products
  “Many languages, One platform”*

• Both have the potential to be highly scalable, available, reliable, etc.

* see www.mono-project.com for open source unix version of .Net framework
Java EE vs .NET

• .NET allows many languages:
  – VB.NET, C#, J#, Managed C++
  – Perl, Python, Eiffel, COBOL
  – & many more

• These languages can invoke objects written in other languages:
  – Common Language Runtime/Common Type System
  – Common Base class libraries
  – eg: C# can instantiate VB.NET class!!

• Can run on Unix via MONO project (beta??)
.NET terminology

- **Assembly** = module to be executed by CLR eg: `.exe` or `.dll` file
  - Nearest equivalent to Java is **JAR** file

- **Attributes** = similar to **Java annotations**, but contained in metadata in assembly

- **ASP.NET** – like **JSP**, but has more (code-behind & event driven web controls)
## .NET versus Java

<table>
<thead>
<tr>
<th>Feature</th>
<th>.NET</th>
<th>Java</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client GUI</td>
<td>WinForms</td>
<td>AWT/Swing</td>
</tr>
<tr>
<td>Data Access</td>
<td>ODBC, ADO.NET</td>
<td>JDBC, JDO</td>
</tr>
<tr>
<td>Directory access</td>
<td>ADSI/ LDAP</td>
<td>JNDI / LDAP</td>
</tr>
<tr>
<td>Remote invocation</td>
<td>.NET remoting</td>
<td>RMI-IIOP</td>
</tr>
</tbody>
</table>
## .NET versus Java EE

<table>
<thead>
<tr>
<th>Feature</th>
<th>ASP.NET</th>
<th>JSP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web GUI</td>
<td>ASP.NET</td>
<td>JSP</td>
</tr>
<tr>
<td>Scripting</td>
<td>ISAPI</td>
<td>Servlet, Filter</td>
</tr>
<tr>
<td>Web server</td>
<td>IIS</td>
<td>Apache, Tomcat ...</td>
</tr>
<tr>
<td>Server side business logic</td>
<td>COM+</td>
<td>Session Beans</td>
</tr>
<tr>
<td>Server-side data logic (manual)</td>
<td>COM+ &amp; ADO.NET</td>
<td>BMP</td>
</tr>
<tr>
<td>Server-side data access component</td>
<td>n/a</td>
<td>CMP</td>
</tr>
<tr>
<td>Directory access</td>
<td>ADSI / LDAP</td>
<td>JNDI / LDAP</td>
</tr>
<tr>
<td>Messaging</td>
<td>MSMQ</td>
<td>JMS</td>
</tr>
<tr>
<td>Transactions</td>
<td>COM+ &amp; DTC</td>
<td>JTA</td>
</tr>
</tbody>
</table>
Main differences

• .NET has no hosting/application server equivalent for EJB

• The **Windows Operating System is the host**
  – (eg: you can run assembly as a “service”, thru IIS)

• Nearest equivalent is “**Component services**” aka **COM+ services**
Java EE vs .NET (2)

- Often the choice of architecture is based on non-technical factors:
  - existing application base
  - cost
  - support

- Web Services are paving the way for both to have equal shares of the market place
  - Interoperability
Summary

• Understand 1, 2, 3, N Tiers and the applicability of each

• Be familiar with some basic design patterns for N Tier applications

• Understand the broad architecture of Java EE
  – we’ll fill in the details later

• Understand that Java EE is not the only option
  – Microsoft .NET is an alternative
  – We’ll stick to Java EE to avoid confusion
Todays Lab

• In today’s lab, you will set up your development environment

• We will use Oracle Weblogic Server 10 for the labs and Oracle Workshop for Weblogic (Eclipse Europa)

• Optionally: You can use Eclipse-all-in-one (Eclipse Ganymede)
Questions?
Appendix

• Distributed Systems lectures from 31284 Web Services Development

• The following explain the different types of Gartner client/server models
Distributed Systems

• Some key characteristics of a distributed system / distributed application include:

  – Resource Sharing
  – Openness
  – Concurrency
  – Scalability
  – Fault Tolerance
  – Transparency
Characteristic: Resource Sharing

- Resource sharing is a fundamental goal of distributed applications
- Enabled by use of architectures we've seen:
  - client/server
  - three-tier
  - n-tier
- Also peer-to-peer (internet file sharing anyone?)
Principle of Sharing

• Resources should be shared so as to minimise load on elements of the distributed system

• Sharing amongst servers
  – Work to be done should be partitioned between servers

• Sharing data
  – Locking must be used to prevent simultaneous modification of shared data
Back to Client/Server ...

• How do you decide whether to do most of the processing at the client, or at the server?

• Fat client vs. thin client
  – Fat client – do lots of work at client, little at server
  – Thin client – do little work at client, lots at server

• This is about distributed applications in general, not specifically web-based applications
  – Web-based apps are mostly thin client
Gartner classification of C/S

Client
- Presentation logic
- Business logic
- Data access

Server
- Presentation logic
- Business logic
- Data access

Distributed data
Remote data
Distributed transaction
Remote presentation
Distributed presentation
notes:

- Here we examine the Garter Group classification of client/server systems into five distinct categories as shown in the slide. The Gartner models are fairly widely known in the distributed computing world, and although getting a bit old now, still illustrate nicely some of the choices that one can make when designing a distributed application.

- The Gartner models range from fat client on the left through to thin client on the right. 'Distributed Transaction' (in the middle) is somewhat of a hybrid. Upcoming slides examine the meaning of each of these models in a little more detail. Note that in the diagrams shown on the slide, the client-side is at the top of the picture, and the server-side is at the bottom.


Gartner: Distributed Data

- The majority of the code runs on the client's computer
- Some of the data is remote, some is local
- e.g.
  - application that accesses one or more distributed databases
notes:

- 'Distributed Data' is a fat-client model. In this case, the majority of the code runs on the client's computer, and the only part of the application that is located remotely is some of the data. But also note that some of the data (or at least, data access code) is also located on the client.

- An example is an application that accesses one or more distributed databases, and possibly stores a local copy of the data as well while working on it. This type of application would be good for "offline" use, e.g. where perhaps the network connection is intermittent (not always available).

- A specific example might be a mobile computing application for data collection, such as meter reading (e.g. water meters, gas meters, etc). A meter reader takes his/her mobile computing device around with them and records the meter reading. However the device is not permanently connected to the network, so throughout the day, the application just stores the data locally. At the end of the day, the meter reader returns to the office, and uploads all of the day's readings in one batch. In this example, all of the presentation logic, business logic and some of the data access is done without using the network. It is just the daily upload that needs the network, and at the server end only data access code is needed (storing data into a database).
Gartner: Remote Data

- The majority of the code runs on the client's computer
- All of the data is remote
- e.g.
  - applications that access data on Networked File Systems
    - NFS on Unix
    - SMB on Windows
notes:

- 'Remote Data' is also a fat-client model. It differs from 'distributed data' in that this time, there is no data or data access code located on the client. This means that all of the data is remote, and must always accessed over the network. This increases the dependency on the network, however does potentially mean that the client computer needs less storage space (as the client is now not storing any data itself).

- An example is networked file system applications, where all of the files are stored on some remote file server. A specific example is logging into a machine in one of the Windows computer labs in the Faculty, and using Microsoft Word to edit a document located on your X: drive. Your X: drive is actually a network drive that is located on a remote file server. All of the presentation and business logic (i.e. Microsoft Word code) is running locally on your client computer. However the data (the file you are editing) is located on a server and accessed via a networked file system (an SMB server in the case of Windows).
Gartner: Distributed Transaction

- The business logic processing is split between the client and the server
  - some business logic at client
  - some business logic at server

- Not a particularly good model – difficulty in updating business logic

- e.g.
  - Gartner example is the web – assuming a Java applet on the client with business logic
'Distributed Transaction' does not neatly fit into either a thin-client or fat-client classification – it is in the middle and a bit of a hybrid. It is distinguished by the fact that some of the business logic runs on the client's computer and some of the business logic runs on the server's computer.

In general this is not a particularly good approach. The business logic of an application is best kept together, so that if the business logic changes, it is (in principle) easier to update if it is all in one place. This is an important consideration, because if an application is going to change, it is more likely to be the business logic than any other part.

Gartner give the example of a web application for the 'distributed transaction' model. The assumption is that the client's computer is executing a Java applet (or an ActiveX control), and that the applet contains some of the business logic of the application. For example, the applet might allow users to enter some data into text fields, and the applet could check that the data entered was correct. It is certainly possible to update both the client-side and server-side business logic, but it would be easier if the business logic was located entirely at the server or entirely at the client, rather than half-half.

Note that generally we think of the web as a thin-client architecture (Gartner's 'distributed presentation' model). It only becomes 'distributed transaction' when a Java applet or ActiveX control is used in the web browser and when that applet/ActiveX contains business logic. "Normal" web applications that only have HTML and JavaScript at the client are 'distributed presentation' in the Gartner classifications.
Gartner: Remote Presentation

- Most of the processing is done at the server, including all business logic

- The server has no concern for how the application will display to the user

  - e.g.
    - web services applications, Java RMI client, vendor applications
Notes:

- ‘Remote Presentation’ is a thin-client model. There is a clean separation between presentation logic (all on the client) and business logic (all on the server). What this means is that the server does not care at all how the presentation is done (i.e. how the application will appear). The server-side of the application just sends data to the client, and the client is responsible for displaying it in an appropriate format.

- Clients could include a “fat” client which uses web services to communicate with the server, or a Java application which invokes a Remote method on a server or even vendor specific applications which communicate via sockets.
Gartner: Distributed Presentation

- Most of the processing is done at the server, including some of the presentation logic

- Both client and server have some responsibility for presentation logic

- e.g.
  - Large web application
    - server side presentation logic could be JSP or PHP
    - client side presentation could be HTML, Javascript that the client-side browser interprets
Notes:

- 'Distributed Presentation' is a thin-client model. It differs from 'remote presentation' in that now the server does have some control over how the application is presented (but the client also still has some control). The presentation logic is split so that some is done at the server and some at the client.

- The most familiar example would be a web application that uses HTML (and possibly JavaScript) for presenting the application to the user. The server is responsible for generating the HTML (e.g. using JSP code). In this sense, the server does have some control over the appearance of the web page – what colours, fonts, etc to use, and how to format the data (paragraphs, tables, etc). However, the client (the web browser) also has some control over the presentation. If a particular font is not available, the web browser will substitute another one. If different users have different screen resolutions, then the web browser will alter the appearance of the web page to best fit the available resolution. The browser takes the HTML instructions from the server, but still has some control in how to carry out those instructions (how to render the page). The client may also run Javascript to do validations and dynamic logic.