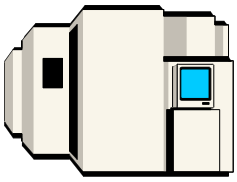
A Comparison of Component Frameworks

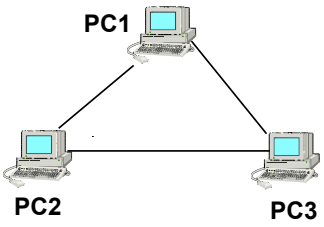
Overview

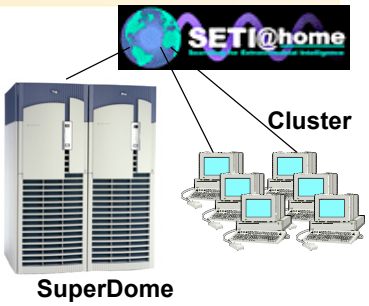
- The Component Landscape
  - Objects vs. Components
  - Concepts of reuse, deployment, (graphical) builder tools
- Component Systems
  - NeXTSTEP Interface Builder, Java Beans, Component Object Model (COM)
- Component Programming with .NET
- Aspects: Problems with Component Frameworks
  - Non-functional component properties
  - Aspect-oriented programming
- How C# and .NET change the scene
  - Dynamic Re-Configuration in a .NET component framework
  - Reflection and attributed-based programming – our aspect weaver
- Conclusions

The Shifting Paradigm

Mainframes → *PC's* → *The Grid*







Hardware →	**Software** →	**Middleware**
IBM →	**Microsoft** →	**???**
closed proprietary →	**closed proprietary** →	**open standards, WebServices**

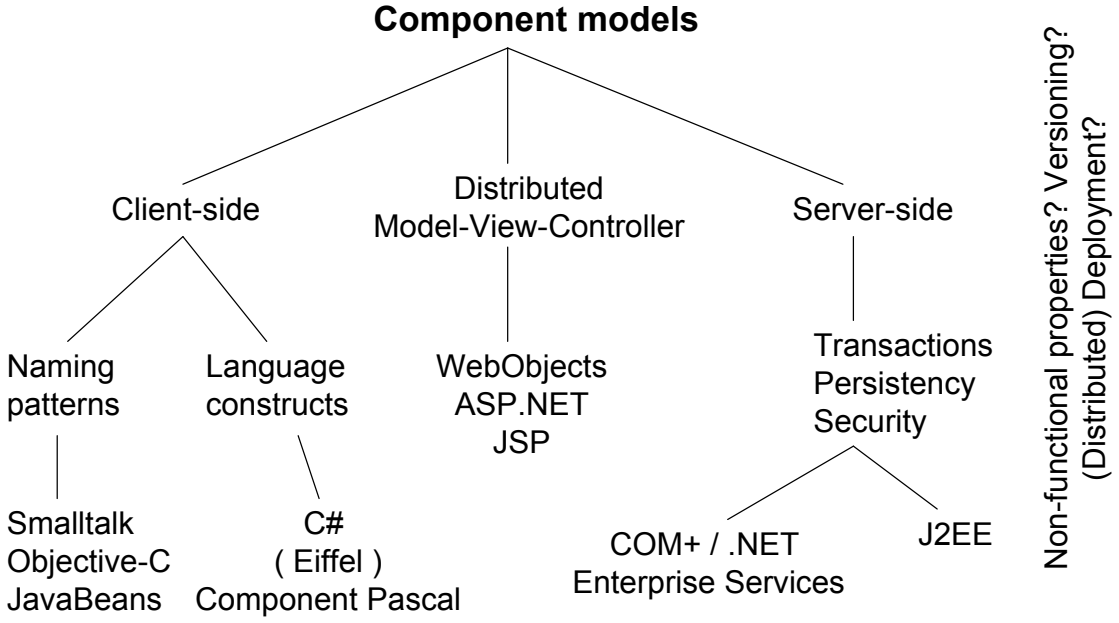
Why Components?

„Software components are binary units of independent production, acquisition, and deployment that interact to form a functioning system" (Szyperski 1997)

The Ultimate Difference

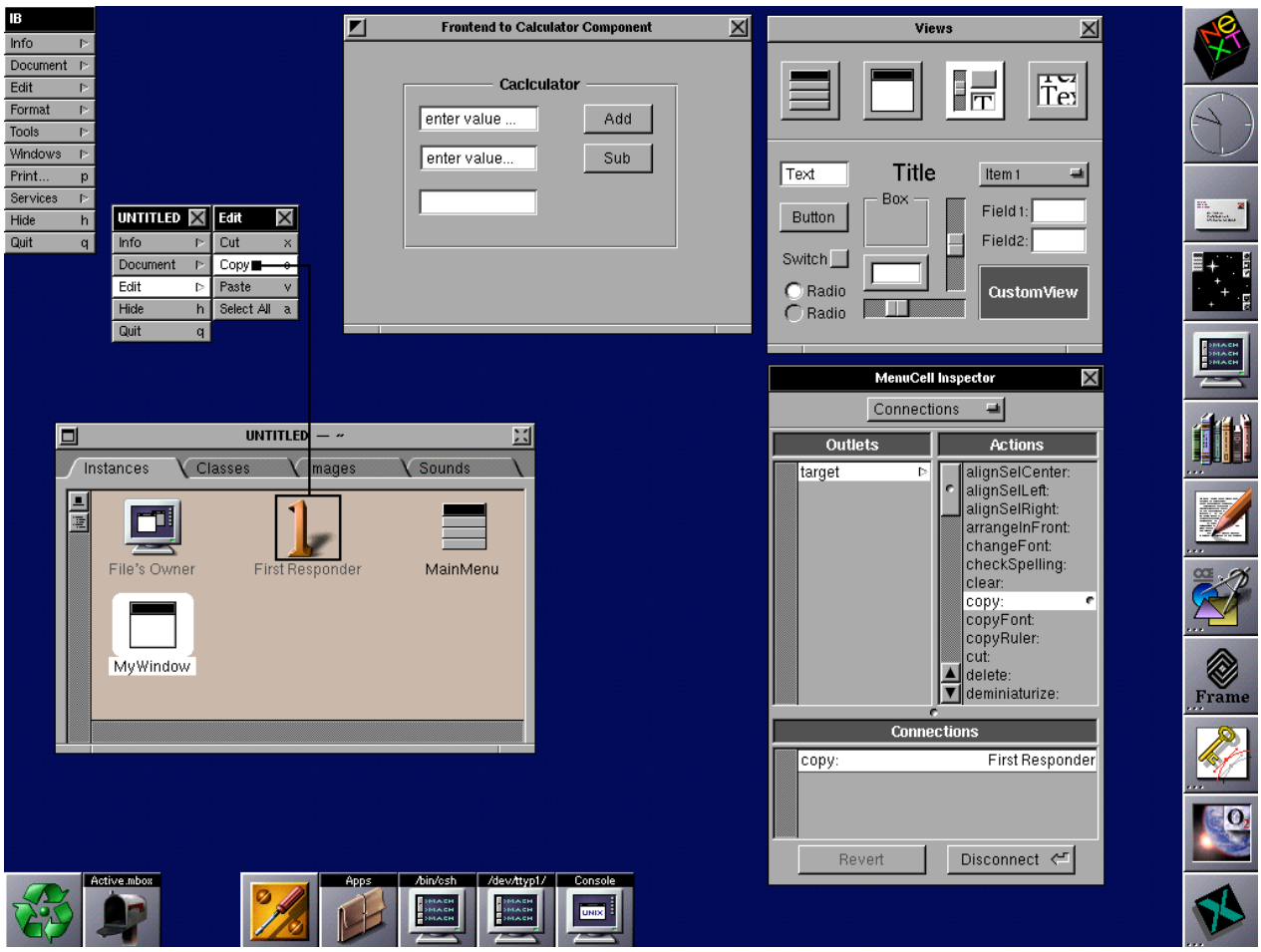
- Components capture the static nature of a software fragment.
- Objects capture its dynamic nature.
  - Simply treating everything as dynamic can eliminate this distinction.
- Good software engineering practices strengthen the static description of systems as much as possible.
  - Dynamics can always be superimposed where needed.
  - Meta-programming and just-in-time compilation simplify this soft treatment of the boundary between static and dynamic.

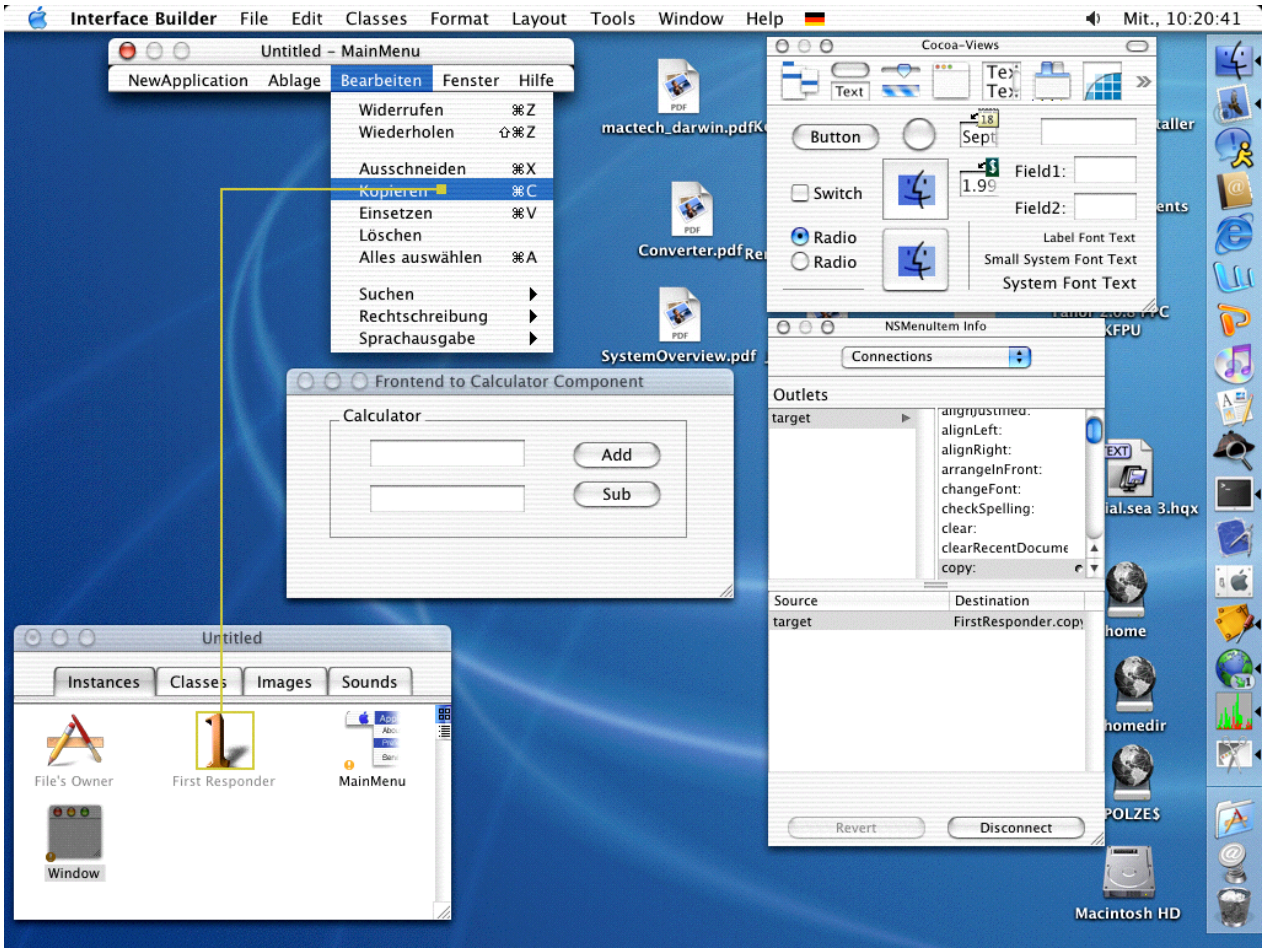
The Component Landscape



I Components at Work

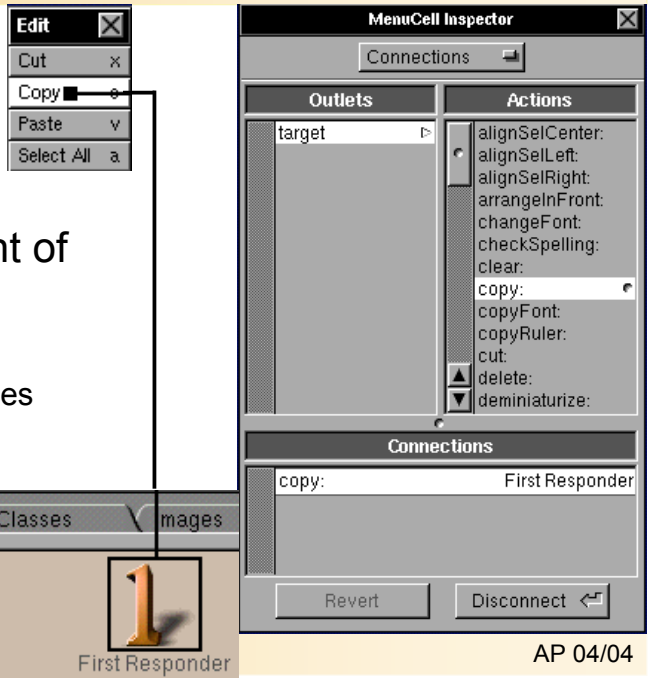
Setting the stage: NeXTSTEP and its tools





NeXTSTEP InterfaceBuilder

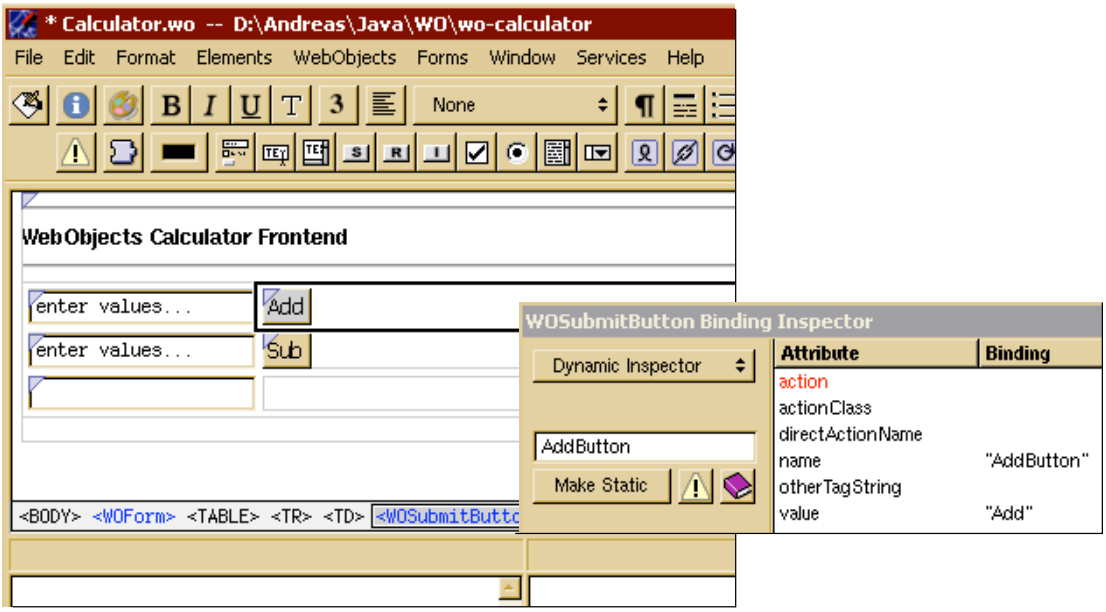
- Component model based on Objective-C
  - Class == object
  - Runtime type info
- Graphical management of events
  - Revolutionary tool (1991)
  - Interconnects event sources and sinks graphically



WebObjects

- Extended component model
  - NeXT's Objective-C classes
  - JavaBeans
- Targeted towards host-based and web-based apps.
  - Generate html-forms
- Implemented on top of OpenStep (for Windows)
  - MacOS X Server, Windows NT/2000
  - All the tools are still available
  - Shift towards Java introduced a number of bugs

WebObjectsBuilder on Windows 2000



II Components at Work

The JavaBeans component model

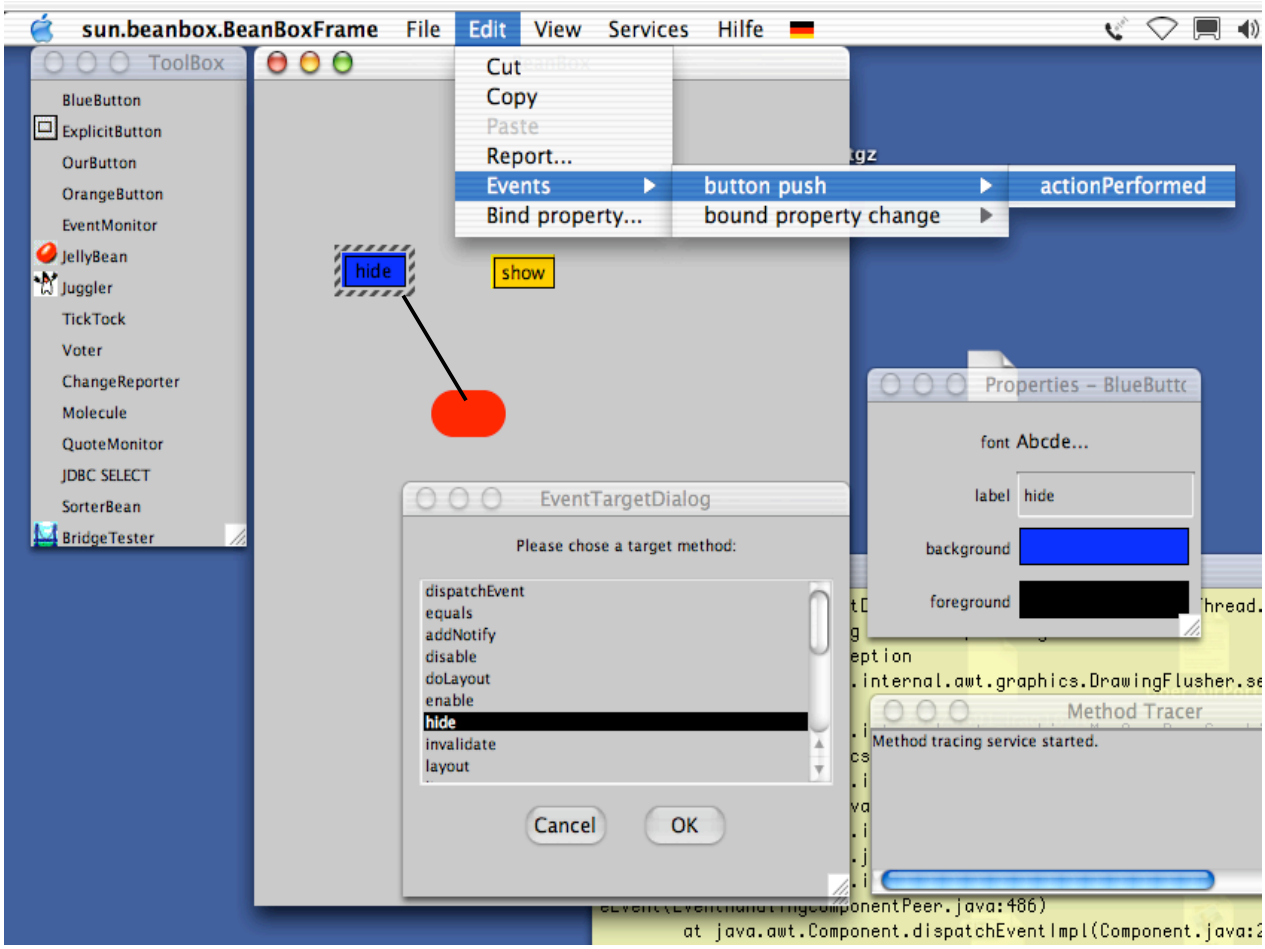
JavaBeans - based on **Naming Patterns**

- Any object that conforms to certain rules can be a bean.
  - No Bean-superclass
- Many beans are AWT components
  - AWT = Abstract Windowing Toolkit
  - Invisible beans may be useful
- A bean is characterized by properties, events, and methods it exports.
- A bean communicates by generating events.
  - Event model is based on java.util.EventObject and java.util.EventListener interfaces

Creating JavaBeans Components

Two rules implied by the **JavaBeans** architecture include:

- the Bean class must provide zero-argument constructors so it can be created using **Beans.instantiate(),** and
- the Bean must support persistence, by implementing either **Serializable** or **Externalizable**.
- Considerations need to be taken for persistence:
  - use of the **transient** keyword when using the default read and write methods.



Methods

- Public methods describe the Bean's behavior.
  - Excluding methods that get/set property values
  - Excluding methods that register/remove event listeners
- Public methods are exposed by the builder tool.
- The methods are used by the builder tool or the user to construct connections between Beans.

Properties

A bean defines a property p of type T if it has an accessor method adhering to the following **naming pattern**:

- Getter:
  - public T getP()
- Boolean getter:
  - public boolean isP()
- Setter:
  - public void setP(T)
- Exceptions:
  - Property accessor methods can throw any type of checked or unchecked exceptions

Special cases:
- Indexed properties
- Bound properties
- Constrained properties
- Property editors
- Customizers

Packaging

- JavaBeans are delivered by means of a **JAR file**.
- A JAR that contains a Bean must include
  - a Manifest declaring the Beans it contains,
  - be it a class or a persistent representation of a prototype of the Bean.
- In the **absence of additional information**, a Bean packaged in a given JAR file may require all the files in that JAR for its successful operation;
  - a builder tool that uses JavaBeans components from a series of JAR files will need to include all files in these JARs

Packaging (contd.)

- Express Inter-Bean dependencies:
  - **Design-Time-Only** and **Depends-On** tags. (JavaBeans 1.01)
- The JAR can also contain any other files
  - such as images, sounds, HTML help files
- The JAR can also contain different localized versions of this information.
  - To support deployment of the Bean in a builder tool the **JavaBeans** component developer can insert **Javadoc** information into the JAR as well.

III

Enterprise Java Beans

Enterprise JavaBeans Technology

- Enterprise JavaBeans (EJB)
  - defines a model for the development and deployment of reusable Java server components.
- Components
  - pre-developed pieces of application code that can be assembled into working application systems.
- JavaBeans
  - support reusable development components.
- EJB architecture
  - logically extends the JavaBeans component model to support server components.

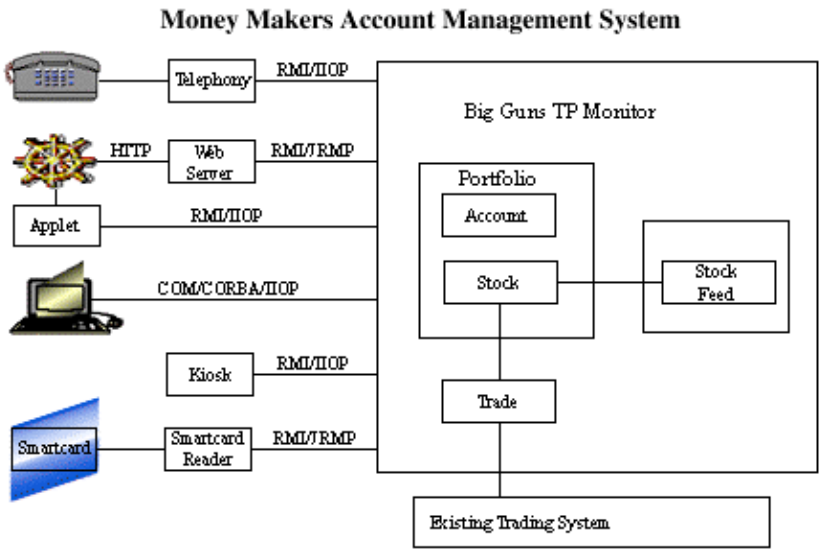
Server Components and Application Servers

- Server components
  - are application components that run in an application server.
  - Applications are based on a multitier, distributed object architecture
  - Most of an application's logic is moved from the client to the server.
  - The application logic is partitioned into one or more business objects that are deployed in an application server.
- Java application server
  - provides an optimized execution environment for server-side Java application components.
  - high-performance, highly scalable, robust execution environment specifically suited to support Internet-enabled application systems.
  - "Write Once, Run Anywhere" (WORA) portability.

Communication Protocols

- Client applications may use a variety of protocols.
  - Java technology clients invoke the application using the native Java Remote Method Invocation (RMI) interface.
  - RMI requests are transferred using the Java Remote Method Protocol (JRMP) or the Internet InterORB Protocol (IIOP).
  - Native language clients can invoke the application using CORBA IDL running over IIOP or a COM/CORBA internetworking service running over IIOP.
  - The RMI client proxy could also be rendered as an ActiveX control to provide easy integration with any Windows application.
  - Browsers can invoke the application through a servlet running on the HTTP server.
  - The browser communicates with the servlet using HTTP, and the servlet communicates with the application using RMI.

Existing Infrastructure Integration



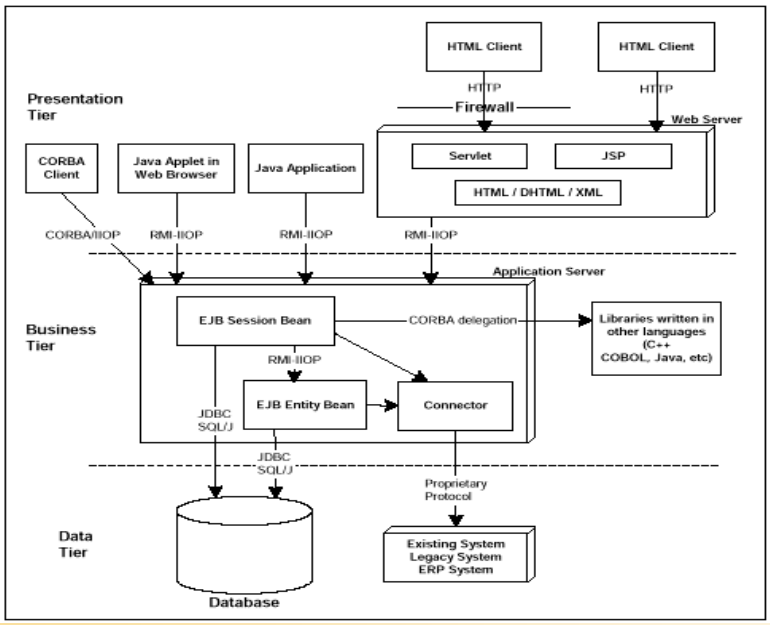
EJB Architecture and their APIs

- EJB
  - The Enterprise JavaBeans API defines a server component model that provides portability across application servers and implements automatic services on behalf of the application components.
- JNDI
  - The Java Naming and Directory Interface API provides access to naming and directory services, such as DNS, NDS, NIS+, LDAP, and COS Naming.
- RMI
  - The Remote Method Invocation API creates remote interfaces for distributed computing on the Java platform.
- Java IDL
  - The Java Interface Definition Language API creates remote interfaces to support CORBA communication in the Java platform. Java IDL includes an IDL compiler and a lightweight, replaceable ORB that supports IIOP.

EJB Architecture and APIs (contd.)

- Servlets and JSP
  - The Java Servlets and Java Server Pages APIs support dynamic HTML generation and session management for browser clients.
- JMS
  - The Java Messaging Service API supports asynchronous communications through various messaging systems, such as reliable queuing and publish-and-subscribe services.
- JTA
  - The Java Transaction API provides a transaction demarcation API.
- JTS
  - The Java Transaction Service API defines a distributed transaction management service based on CORBA Object Transaction Service.
- JDBCTM
  - The JDBC Database Access API provides uniform access to relational databases, such as DB2, Informix, Oracle, SQL Server, and Sybase.

EJB Object Model



Enterprise JavaBeans Component Model

- The Enterprise JavaBeans component model logically extends the JavaBeans component model to support server components.
- Server components
  - are reusable, prepackaged pieces of application functionality that are designed to run in an application server.
  - They can be combined with other components to create customized application systems.
- Server components
  - are similar to development components, but they are generally larger grained and more complete than development components.
- Enterprise JavaBeans components (enterprise beans)
  - cannot be manipulated by a visual Java IDE in the same way that JavaBeans components can.
  - Instead, they can be assembled and customized at deployment time using tools provided by an EJB-compliant Java application server.

Implicit Services

- The EJB container performs a number of service on behalf of the enterprise beans-
- Lifecycle.
  - Individual enterprise beans do not need to explicitly manage process allocation, thread management, object activation, or object destruction.
- State Management.
  - Individual enterprise beans do not need to explicitly save or restore conversational object state between method calls.
- Security.
  - Individual enterprise beans do not need to explicitly authenticate users or check authorization levels.

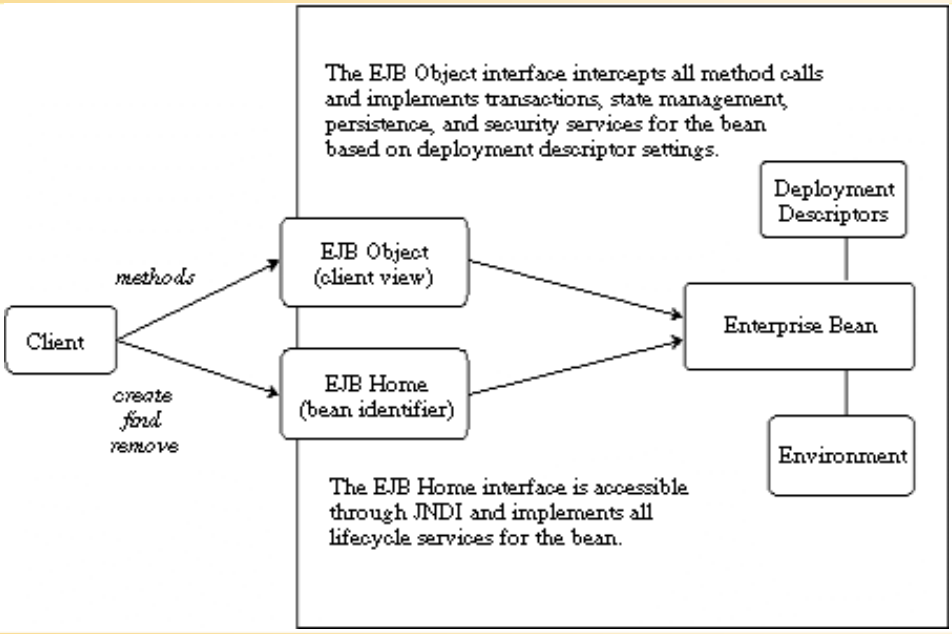
Implicit Services (contd.)

- Transactions.
  - Individual enterprise beans do not need to explicitly specify transaction demarcation code to participate in distributed transactions.
  - The EJB container can automatically manage the start, enrollment, commitment, and rollback of transactions on behalf of the enterprise bean.
- Persistence.
  - Individual enterprise beans do not need to explicitly retrieve or store persistent object data from a database.
  - The EJB container can automatically manage persistent data on behalf of the enterprise bean.

Potential Enterprise JavaBeans Systems

- TP monitors,
  - such as IBM TXSeries and IBM CICS/390
- Component transaction servers,
  - such as Sybase Jaguar CTS
- CORBA systems,
  - BEA Systems M3, IBM WebSphere Advanced Edition, Inprise VisiBroker/ITS
- Relational database systems,
  - such as IBM DB2, Informix, Oracle, and Sybase
- Object database systems,
  - such as GemStone GemStone/J
- Object/relational caching systems,
  - Persistence PowerTier and Secant Extreme
- Web application servers,
  - BEA WebLogic, Bluestone Sapphire, IBM WebSphere, Netscape Application Server, Oracle Application Server, Progress Apptivity, SilverStream Application Server, and Sun NetDynamics.

EJB Container



Session Beans

- A session bean is created by a client
  - exists only for the duration of a single client/server session.
- Performs operations on behalf of the client
  - such as accessing a database or performing calculations.
- Can be transactional
  - but (normally) they are not recoverable following a system crash.
- Can be stateless
  - or they can maintain conversational state across methods and transactions.
  - The container manages the conversational state of a session bean if it needs to be evicted from memory.
- A session bean must manage its own persistent data.

Entity Beans

- Object representation of persistent data
  - maintained in a permanent data store, such as a database.
  - A primary key identifies each instance of an entity bean.
- Entity beans can be created
  - either by inserting data directly into the database or by
  - creating an object (using an object factory Create method).
  - Entity beans are transactional, and they are recoverable following a system crash.
- Support for session beans is required,
  - but support for entity beans and container-managed persistence is optional.

Enterprise JavaBeans Security

- Automates the use of Java platform security
  - enterprise beans do not need to explicitly code Java security routines.
  - The security rules for each enterprise bean are defined declaratively in a set of AccessControlEntry objects within the deployment descriptor object.
  - An AccessControlEntry object associates a method with a list of users that have rights to invoke the method.
  - The EJB container uses the AccessControlEntry to automatically perform all security checking on behalf of the enterprise bean.

Packaging

- EJB components can be packaged
  - as individual enterprise beans,
  - as a collection of enterprise beans, or
  - as a complete application system.
- EJB components are distributed in a Java Archive File
  - called an ejb-jar file.
  - The ejb-jar file contains a manifest file outlining the contents of the file,
  - plus the enterprise bean class files,
  - the Deployment Descriptor objects, and, optionally,
  - the Environment Properties objects.

Deployment

- Deployment Descriptor object
  - used to establish the runtime service settings for an enterprise bean.
  - tells the EJB container how to manage and control the enterprise bean.
  - The settings can be set at application assembly or application deployment time.
- Deployment Descriptor defines
  - the enterprise bean class name,
  - the JNDI namespace that represents the container,
  - the Home interface name,
  - the Remote interface name, and
  - the Environment Properties object name.

IV

The Component Object Model (COM+)

COM – The idea

- COM is based on three fundamental ideas
- Clients program in terms of interfaces, not classes (classic COM)
- Implementation code is not statically linked, but rather loaded on-demand at runtime (classic COM)
- Object implementers declare their runtime requirements and the system ensures that these requirements are met (MTS & COM+)

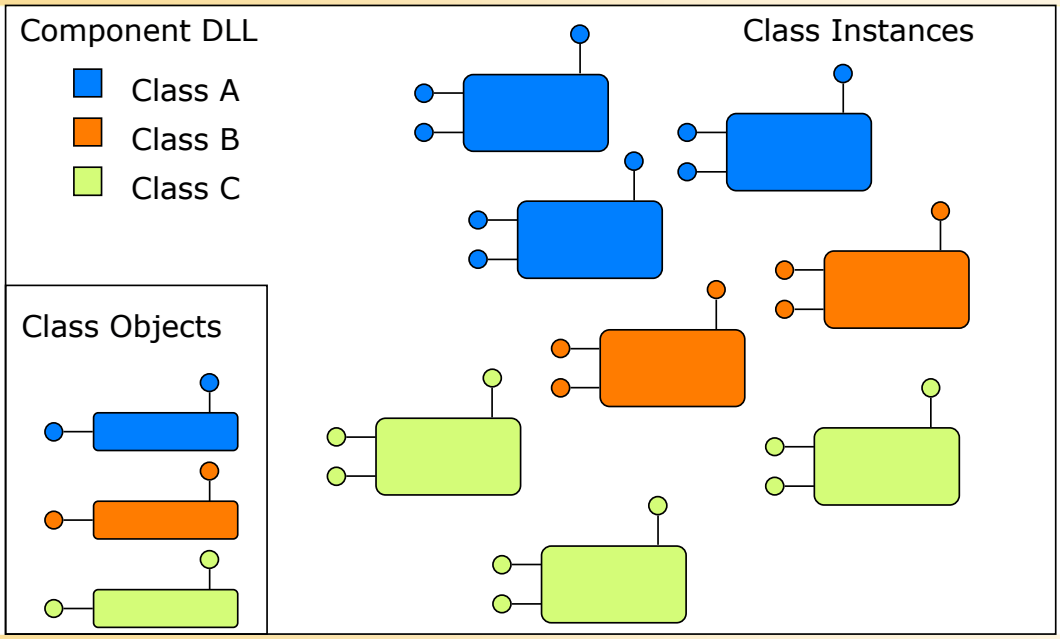
Definitions

- Two key terms have been defined so far
- A COM Interface is a collection of abstract operations one can perform on an object
  - Must extend IUnknown directly or indirectly
  - Identified by a UUID (IID)
  - Platform-specific vptr/vtable layout
- A COM Object is a collection of vptrs in memory that follow the COM identity laws
  - Must implement at least one COM interface
  - QueryInterface ties vptrs together into cohesive object
- Objects assumed to materialize from thin air!

Definitions

- A COM Class (or coclass) is a named body of code that can be used to produce COM objects
- All coclasses are named by a UUID (CLSID)
- All coclasses have a distinguished object that is used to create new instances
  - Called a class object or class factory
  - Typically implements IClassFactory
- All coclasses loaded on demand by class loader
  - Called the Service Control Manager or (SCM)
- For efficiency, a single component DLL can support multiple COM classes

Classes, Class Objects, And Components



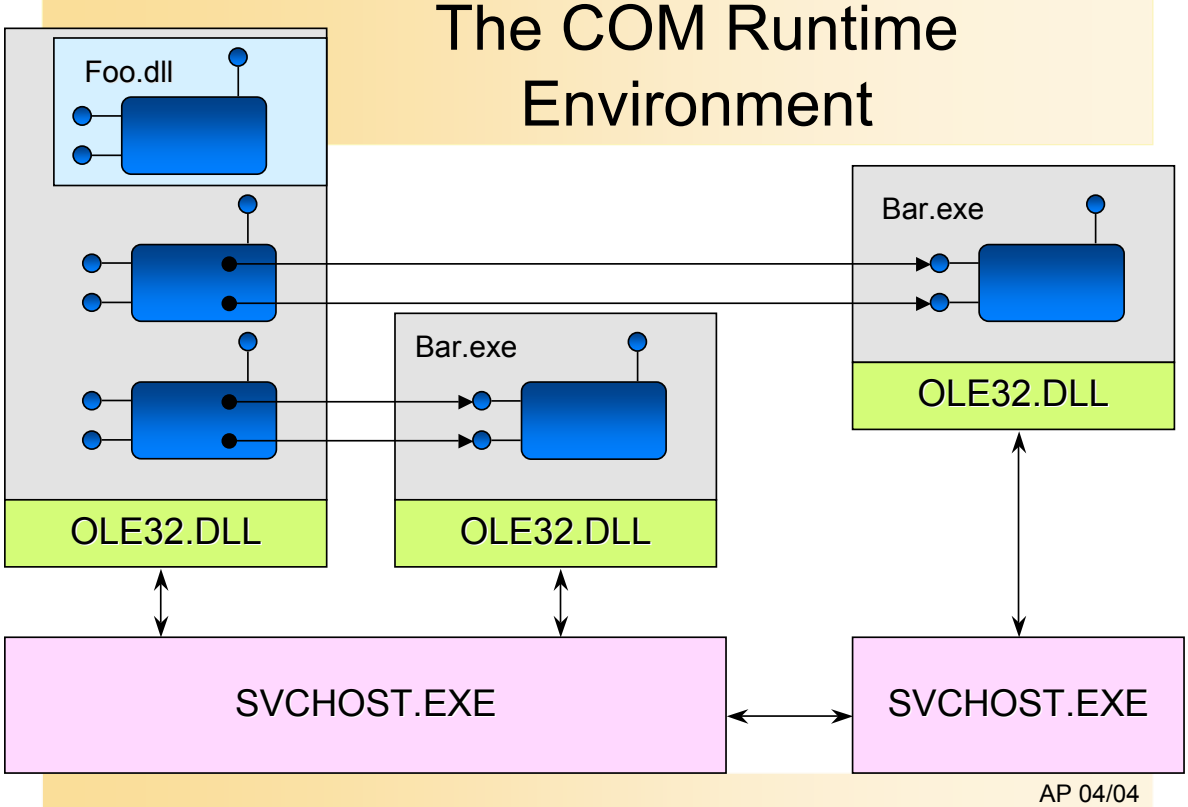
Class Versus Type

- An Interface represents a data type suitable for declaring variables
  - Non-trivial operations
  - Hierarchical with respect to one another
  - Polymorphic with respect to different objects
- A Class represents loadable concrete code used to create objects
  - Resultant objects implement one or more interfaces
- Class unsuitable for declaring variables
  - Entire motivation for interface-based programming based on relative uselessness of class

The COM+ Runtime Environment

- The infrastructure used to support COM+ on a given platform is called the COM+ library
- Each thread that uses COM+ library must setup/teardown thread-specific data structures
  - CoInitialize[Ex] and CoUninitialize do this for you
- The COM+ library implemented in several DLLs
  - OLE32.DLL – core class/interface functionality
  - OLEAUT32.DLL – Visual Basic®-centric type infrastructure
- Inproc class loading done in OLE32.DLL
- Cross-process/host class loading performed by Windows 2000 Service (SVCHOST.EXE)

The COM Runtime Environment



COM Class Loading

- Clients issue activation calls against the Service Control Manager (SCM)
- SCM responsible for locating component and loading it into memory
- SCM queries component for class object and (optionally) uses it to instantiate new instance
- Once SCM returns a reference to class instance/class object, SCM out of the picture
- Based on configuration, COM may need to load component in separate process (potentially on different machine)

COM Class Loading And Locality

- All activation calls allow client to indicate locality
  - SCM chooses most efficient allowed by client

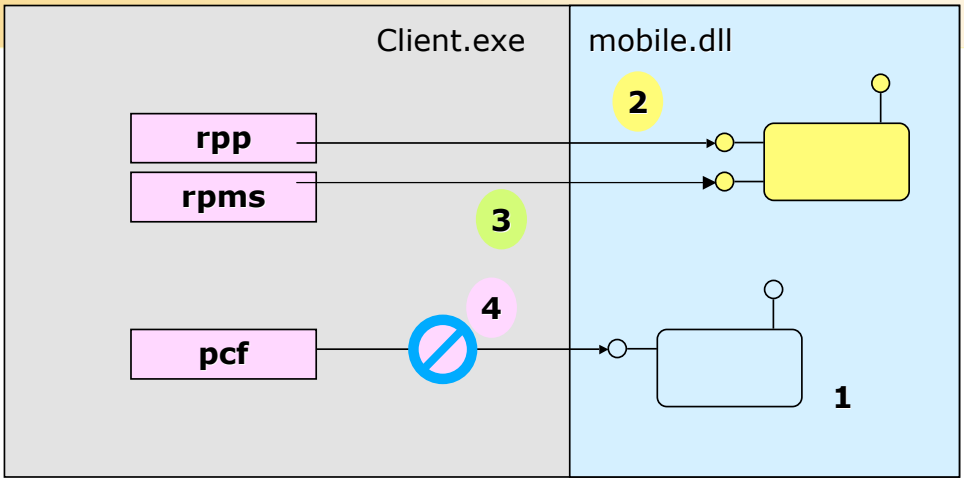
```
CLSCTX_INPROC_SERVER  // load in client process
CLSCTX_INPROC_HANDLER // use OLE Document rendering handler
CLSCTX_LOCAL_SERVER   // load in separate process
CLSCTX_REMOTE_SERVER  // load on distinct host machine
CLSCTX_SERVER         // CLSCTX_*_SERVER
CLSCTX_ALL            // CLSCTX_*
```

```
typedef struct _COSERVERINFO {
  DWORD dwReserved1;       // m.b.z.
  const OLECHAR *pwszName; // host name
  COAUTHINFO *pAuthInfo;   // security goo
  DWORD dwReserved2;       // m.b.z.
} COSERVERINFO;
```

Using The SCM

- The SCM exposes two core activation APIs
- Both APIs load component automatically
- Both APIs accept a CLSID and information about component location as input parameters
- **CoGetClassObject** returns class object/factory
  - No new instances created
- **CoCreateInstanceEx** uses **IClassFactory** interface on class object to create new instance
  - Class object never returned to client

Example



1 Client calls CoGetClassObject

2 Client calls CreateInstance on Class Object

3 Client calls QueryInterface on Class Instance

4 Client calls Release on Class Object

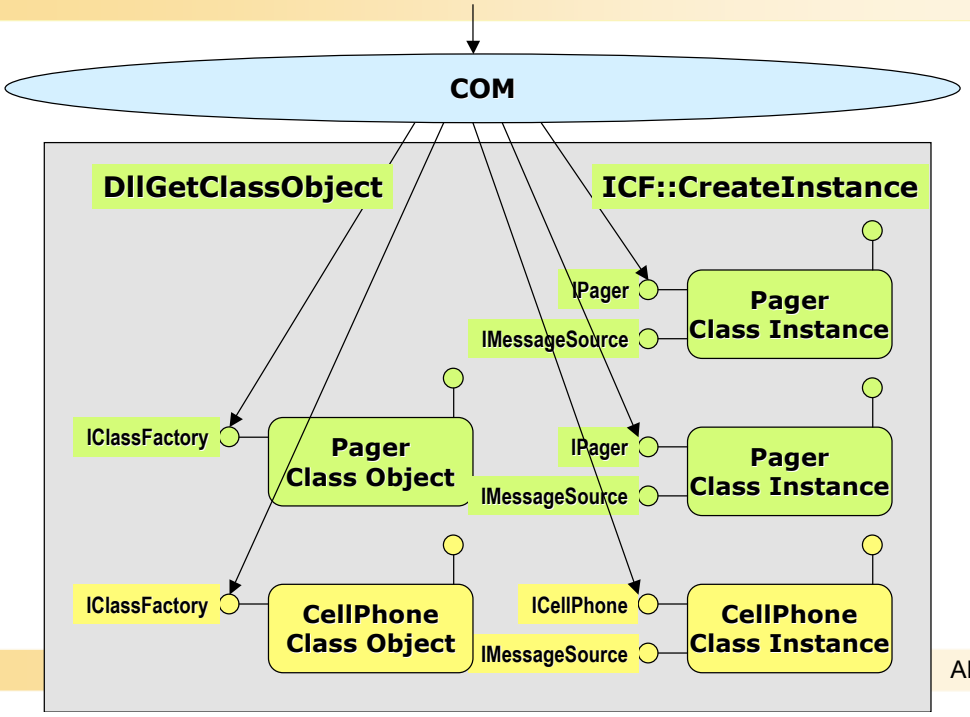
Exposing COM Classes

- Component DLLs export a well-known function to extract class object from DLL

```
STDAPI DllGetClassObject(
        [in] REFCLSID rclsid, // which class?
        [in] REFIID   riid,   // which interface?
        [out, iid_is(riid)] void **ppv // put it here!
);
```

- **DllGetClassObject** is called by **CoGetClassObject** and **CoCreateInstance[Ex]** to access class object
  - Never called directly by client code
- If DLL doesn't export DllGetClassObject, all activation calls will fail

DllGetClassObject



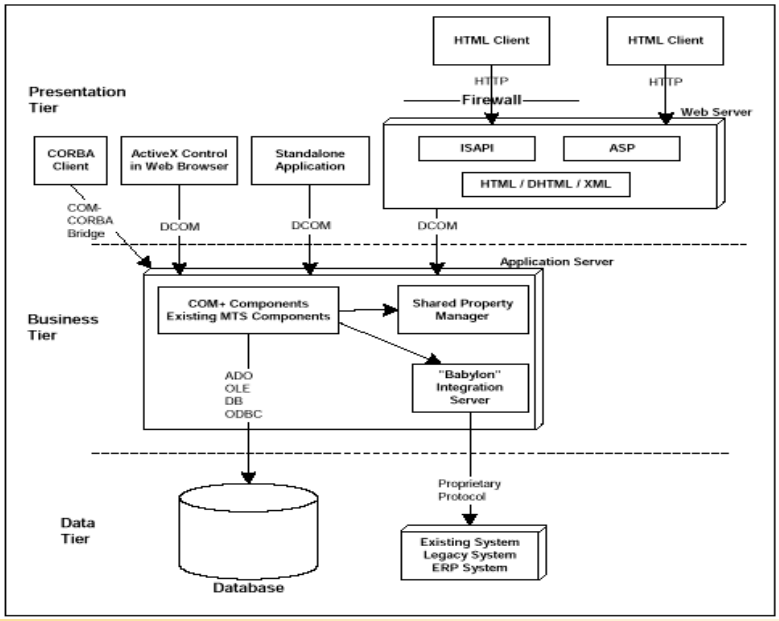
Finding Components

- All COM classes registered under distinguished key in registry (HKEY_CLASSES_ROOT)
  - Holds machine-wide configuration under Windows NT 4.0
  - Magic key under W2K that merges machine-wide registration with current user's private configuration
- Can also register text-based aliases for CLSIDs called ProgIDs for GUID-hostile environments
- REGSVR32.EXE used to install component DLLs that export two well-known entry points
  - STDAPI DllRegisterServer(void);
  - STDAPI DllUnregisterServer(void);

MTS – EJB Competition

- Although Microsoft Transaction Server (MTS) could be adapted to support Enterprise JavaBeans components, Microsoft is not likely to make the effort.
- MTS provides a container system for COM server components, providing transactional and security services similar to those provided in Enterprise JavaBeans servers.
- COM+, the next generation of MTS, will provide a few additional capabilities, such as dynamic load-balancing and queued request-processing.

Microsoft Windows DNA Object Model



Conclusions

- NeXT‘s beautiful Objective-C classes are gone
- The JavaBeans component model brings
  - Programming rules and naming conventions
  - Java language-binding only, no distribution of components
- Microsoft Component Object Model (COM+)
  - Mature distributed component framework
  - Separate type system and multiple language bindings
  - Based on binary memory layout of C++ vtables (!)
- End-of-last-century style of component programming
  - Non-functional component properties – Deployment, Versioning, FT, RT, Configuration are insufficiently addressed

V

Components in C# and .NET

What defines a component?

- What defines a component?
  - Properties, methods, events
  - Design-time and runtime information
  - Integrated help and documentation
- C# has first class component support
  - Not naming patterns, adapters, etc.
  - Not external files
  - Versionable
- Easy to build and consume

Properties

- Properties are “smart fields”
  - Natural syntax, accessors, inlining

```
public class Button: Control
{
   private string caption;

   public string Caption {
      get {
         return caption;
      }
      set {
         caption = value;
         Repaint();
      }
   }
}
```

```
Button b = new Button();
b.Caption = "OK";
String s = b.Caption;
```


Indexers

- Indexers are “smart arrays”
  - Can be overloaded

```
public class ListBox: Control
{
   private string[] items;

   public string this[int index]{
      get {
         return items[index];
      }
      set {
         items[index] = value;
         Repaint();
      }
   }
}
```

```
ListBox listBox = new ListBox();
listBox[0] = "hello";
Console.WriteLine(listBox[0]);
```

Events

- Efficient, type-safe and customizable
  - Built on delegates

```
public class MyForm: Form {
  public MyForm()
  {
    Button okButton = new Button(...);

    okButton.Click += new EventHandler(OkButtonClick);
  }

  void OkButtonClick(…)
  {
    ShowMessage("You clicked OK");
  }
}
```

Design/Runtime Information

- Add information to types + methods?
  - Transaction required for a method?
  - Mark members for persistence?
  - Default event hookup?
- Traditional solutions
  - Lots of custom user code
  - Naming conventions between classes
  - External files (e.g. .IDL, .DEF)
- The C# solution - Attributes

Attributes

- Appear in square brackets
- Attached to code elements

```
[HelpUrl("http://SomeUrl/Docs/SomeClass")]
class SomeClass
{
   [WebMethod]
   void GetCustomers() { … }

   string Test([SomeAttr] string param1) {…}
}
```


Attribute Fundamentals

- Attributes are classes! Completely generic

```
class HelpUrl : System.Attribute {
    public HelpUrl(string url) { … }
    …
}
```

```
[HelpUrl("http://SomeUrl/APIDocs/SomeClass")]
class SomeClass { … }
```

- Easy to attach to types and members

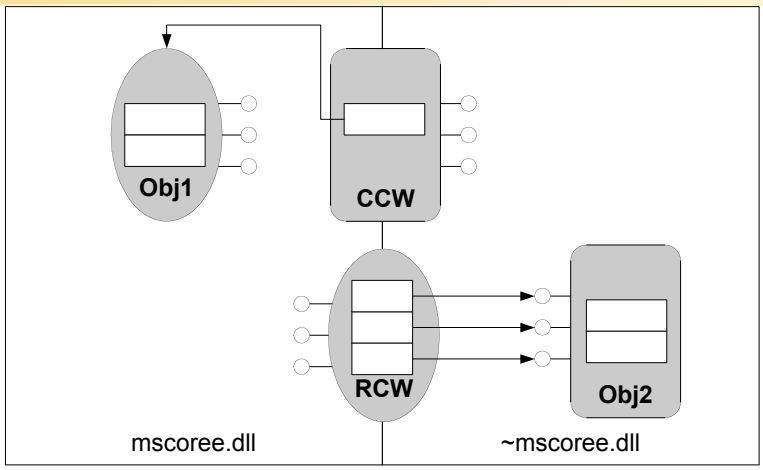
```
Type type = Type.GetType("SomeClass");
object[] attributes =
     type.GetCustomAttributes();
```

- Attributes can be queried at runtime

COM Support

- .NET Framework provides great COM support
  - TLBIMP imports existing COM classes
  - TLBEXP exports .NET types
- Seamless experience

on-the-fly COM



- COM callable wrapper (CCW)
- runtime callable wrapper (RCW)

Calling into a COM component

- Create .NET assembly from COM component via **tlbimp**
- Client apps may access the newly created assembly

```
using System;
using System.Runtime.InteropServices;

using CONVERTERLib;

class Convert {
        public static void Main(string [] args) {
        CFConvert conv = new CFConvert();
        ...
        fahrenheit = conv.CelsiusToFahrenheit( celsius );
```


.NET Programming in Managed C++

CLR operation	C#	Managed C++
Interface declaration	interface IFoo{}	__gc __interface IFoo{}
Class declaration	class Foo{}	__gc class Foo{}
Property declaration	int x {get; set;}	__property int get_x(); __property void set_x(int x);

Property impl.	int x {
	get{ return m_;}
	set{ m_x = x; }}
		__property int get_x() {
		return m_x; }
		__property void set_x(int x)
		{
		m_x = x; }

Interface impl.	class Foo: IFoo {}	class Foo: public IFoo {}

Managed C++ (contd.)

CLR operation	C#	Managed C++
Delegate declaration	delegate void call();	__delegate void call();
Indexer declaration	String this[int index] { .... }	[System::Reflection::DefaultMemberAttribute(S“Item“)] __gc class MyCollection { __property String* get_Item(int index); }
Assembly reference	/r:assembly.dll	#using <assembly.dll>
Namespace import	using System;	using namespace System;
Namespace ref.	System.Console.WriteLine(„text“);	System::Console::WriteLine(„text“);

Managed C++ (contd.)

CLR operation	C#	Managed C++
Object variable	IFoo foo = new Foo();	IFoo * pFoo = new Foo();
Method invocation	foo.DoFoo();	pFoo->DoFoo();
Enumeration decl.	enum Foo { bar, qx }	__value enum Foo { bar, qx }
Name clash resolution	void IArtist.Draw() {} void ICowBoy.Draw(){}	void IArtist::Draw() {} void ICowBay.Draw() {}
Value type	struct Foo { ... }	__value struct Foo{ ... };
Abstract type	abstract class Foo { .. }	__abstract class Foo {.. };
Sealed type	sealed class Foo { ... }	__sealed class Foo { ... };

Managed C++ (contd.)

CLR operation	C#	Managed C++
C-style typecast	try { IFoo foo = (IFoo)bar; } catch (IllegalCastException e){}	try { IFoo* pFoo = __try_cast<IFoo*>(pBar); } catch (IllegalCastException* e){}
Dynamic type cast	IFoo foo = bar as IFoo; if (foo != null) ...	IFoo* pFoo = dynamic_cast<IFoo*> (pBar); if (pFoo != NULL) ...
Type check	if (bar is IFoo) ...	if (dynamic_cast<IFoo*> (pBar) != NULL) ...

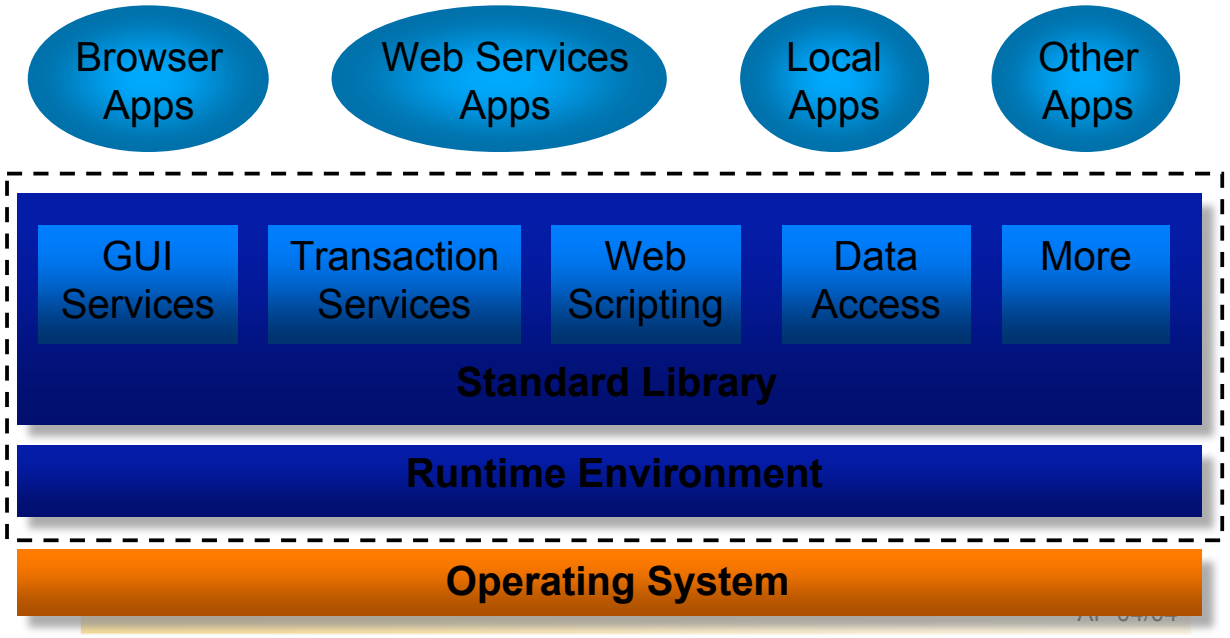
VI

Comparison
J2EE vs. .NET

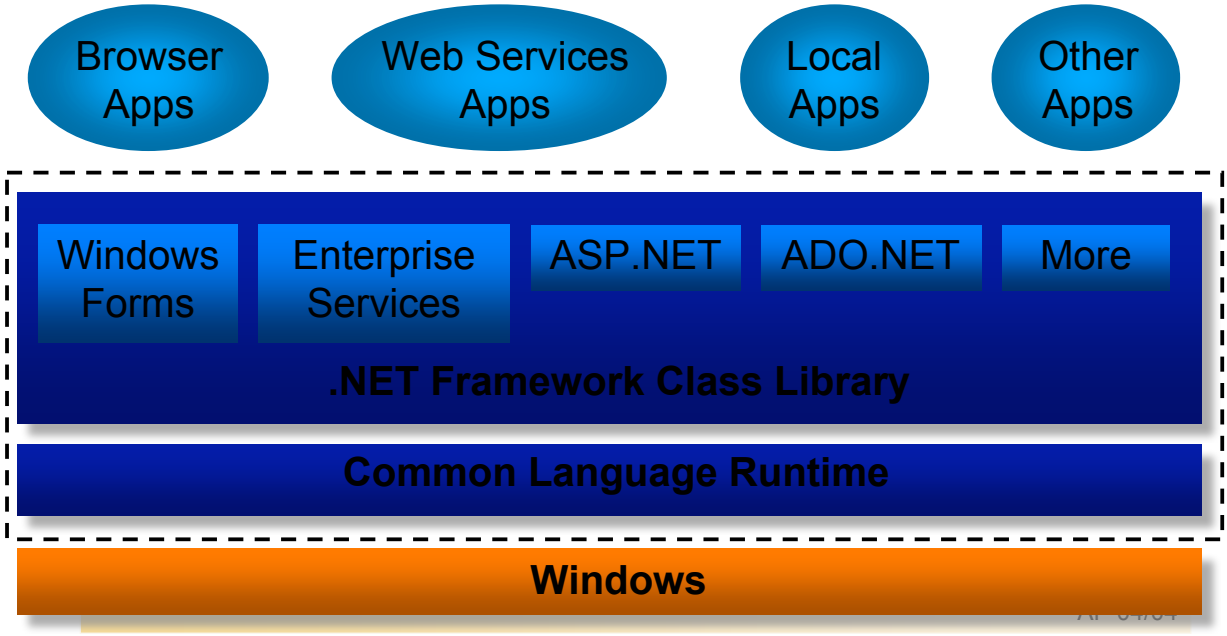
Areas for Comparison: Application Platforms

- .NET
  - The .NET Framework
- Java
  - Java application servers
  - Products include:
    - IBM WebSphere Application Server
    - BEA WebLogic Application Server
    - Sun ONE Application Server
    - Oracle Application Server
    - Many others

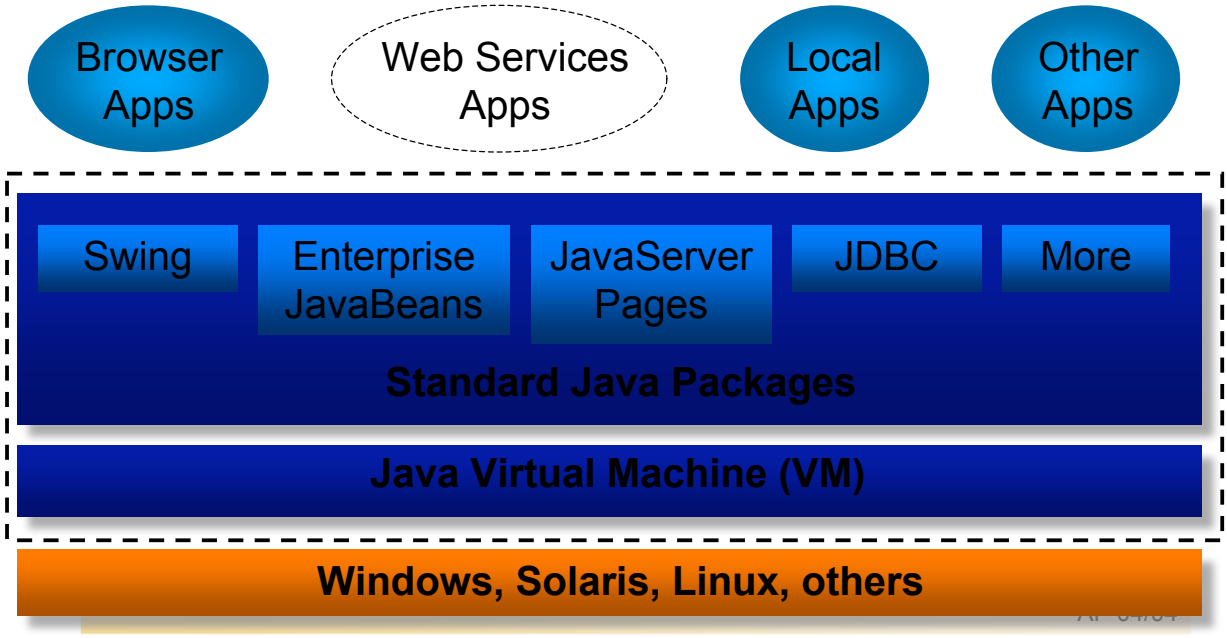
Application Platforms Today



Illustrating The .NET Framework



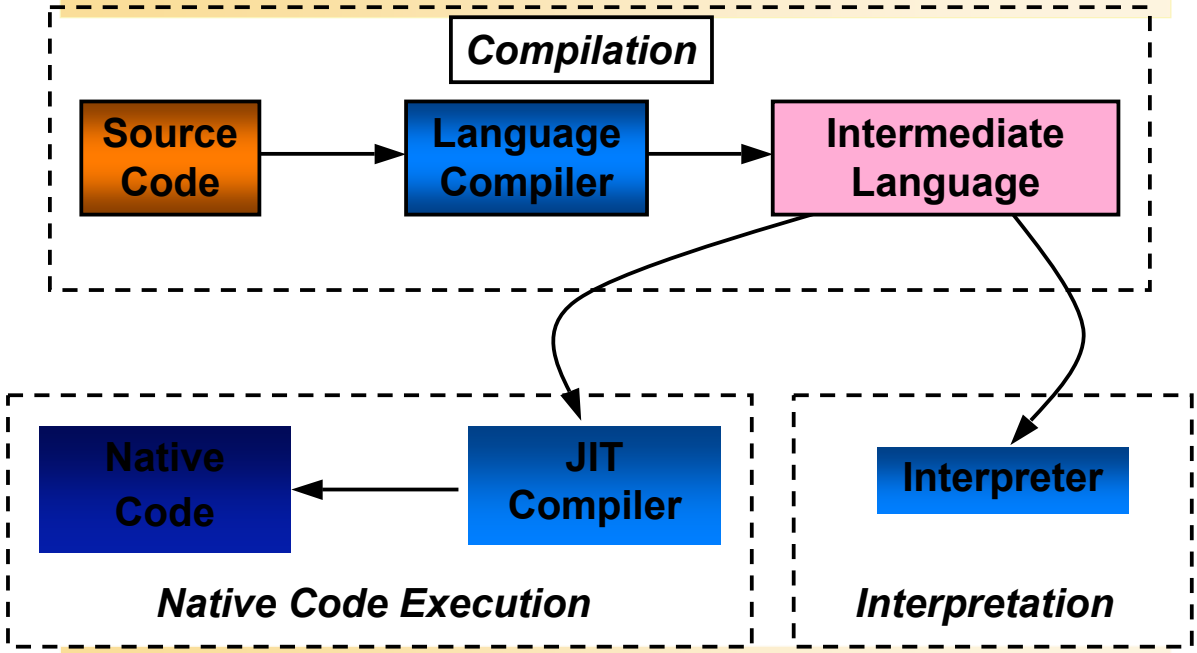
Illustrating The Java Environment



Application Platforms: Some History

	1996	1998	2002
Microsoft	Windows DNA - *MTS (now COM+)* - *ASP* - *ADO*		.NET Framework - *CLR* - *C#, VB.NET* - *Enterprise services, ASP.NET, ADO.NET* - *Web services support*
Java	Java - *Java language* - *Java VM* - *J2SE*	J2EE - *EJB* - *JSP* - *JDBC*	

Runtime Environments Today



Areas for Comparison: Runtime Environments

- .NET Framework
  - Intermediate language is Microsoft Intermediate Language (MSIL)
  - Provides JIT compilation
    - There is no interpreter in the CLR
- Java
  - Intermediate language is bytecode
  - Original implementation targeted interpretation
    - Java VMs with JIT compilation are now also used

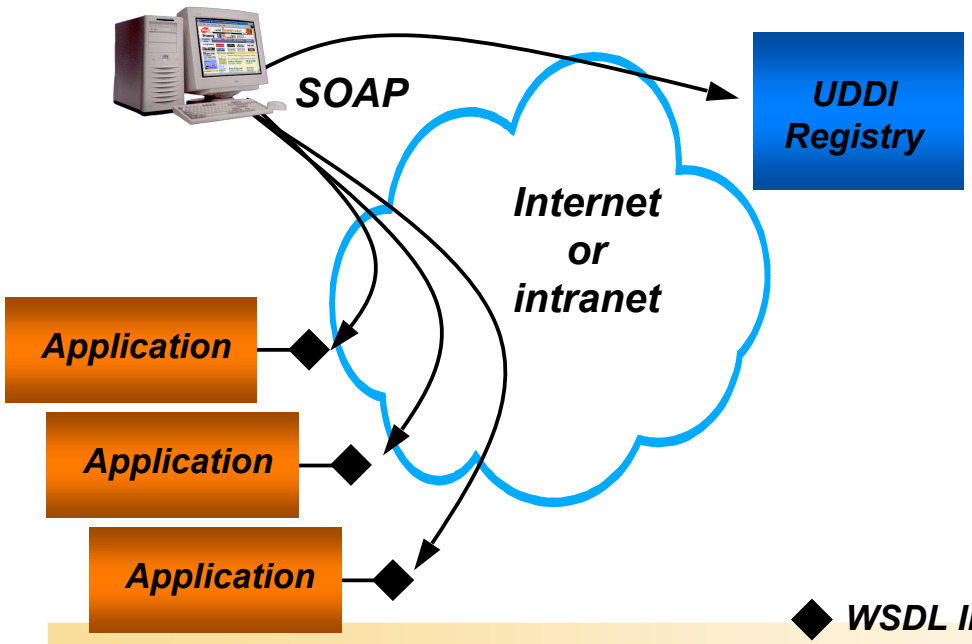
Defining Web Services

- A web service is a chunk of functionality that can be accessed using web technology
  - Across an intranet or the Internet
- Usually:
  - What's accessed is a method in a language such as Java or Visual Basic.NET
  - The web technology used is the Simple Object Access Protocol (SOAP) carried over HTTP
- Both the .NET and Java camps have endorsed web services

Web Services Technologies

- Simple Object Access Protocol (SOAP)
  - An XML-based way to invoke web services
- Web Services Description Language (WSDL)
  - An XML-based language for describing web services
- Universal Description, Discovery, and Integration (UDDI)
  - A registry for web services
- Global XML Web Services Architecture (GXA)
  - Addressing security and much more

Illustrating Web Services Technologies



Applying Web Services

- Connecting clients to applications
- B2B integration on the Internet
- Enterprise application integration (EAI) on intranets

Web Services Support

- .NET Framework class library
  - System.Web.Services namespace
    - Implements ASP.NET's .asmx pages
  - System.Runtime.Remoting namespace
    - Provides SOAP support for remote access
- J2EE 1.3
  - Web services support will be added in the next version
  - Although some vendors, e.g., IBM, have added it on their own

Summarizing the Technologies (1)

	.NET	Java
Application Server	.NET Framework	IBM WebSphere, BEA WebLogic, others
Runtime Environment	Common Language Runtime (CLR)	Java Virtual Machine (VM)
Standard Libraries	.NET Framework class library	J2SE, J2EE
GUIs	Windows Forms	Swing
Transactions	Enterprise Services	EJB

Summarizing the Technologies (2)

	.NET	Java
Web Scripting	ASP.NET	JSPs
Data Access	ADO.NET	JDBC
Small Device Platform	.NET Compact Framework	J2ME
Development Tools	Visual Studio.NET	IBM WebSphere Studio, Borland JBuilder, others
Web Services Support	ASP.NET, .NET My Services, others	Some support from IBM, et al., Liberty Alliance

Areas for Comparison: Non-Technical Issues

- Vendor issues
  - Lock-in
  - Trust
- Maturity
- Existing developer skills
- Operating system support
- Cost

Conclusions

- The development world has bifurcated
  - Microsoft .NET
  - The Java environment
- Both have similar architectures
- Both will survive
  - Which is a good thing
- .NET will dominate in the Windows environment