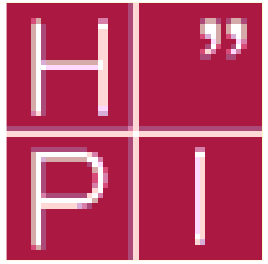


Component Programming with J2EE and .NET

Prof. Dr. Andreas Polze

Operating Systems and Middleware Chair
Hasso-Plattner-Institute
at University Potsdam

Prof.-Dr.-Helmert-Str. 2-3
14482 Potsdam, Germany

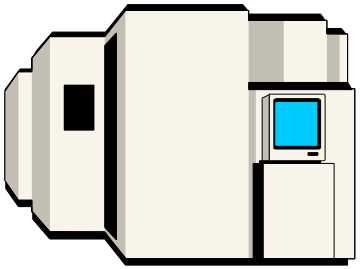
andreas@polze.de
andreas.polze.de, www.dcl.hpi.uni-potsdam.de

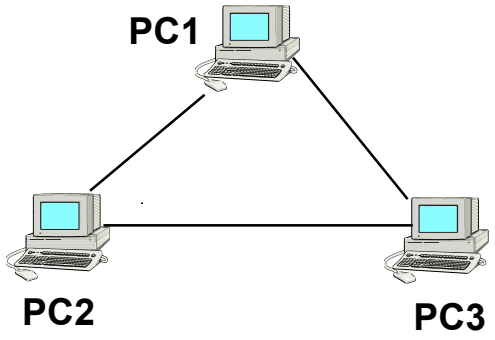
Overview

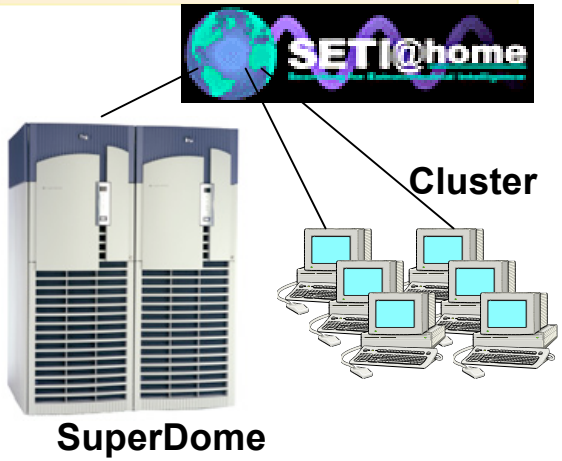
- The Notion of Component-Oriented Programming
  - Objects vs. Components
  - Concepts of reuse, deployment, (graphical) builder tools
- Component Systems
  - NeXTSTEP Interface Builder, Java Beans, Component Object Model (COM)
- **Areas of Comparison**
  - **Client-side: JavaBeans vs. C# constructs**
  - **Server-side: EJB architecture vs. MTS/COM+ Enterprise services**
  - **Summary of technologies**
- Aspects: Problems with Component Frameworks
  - Non-functional component properties - Aspect-oriented programming
  - How C# and .NET change the scene
- 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)

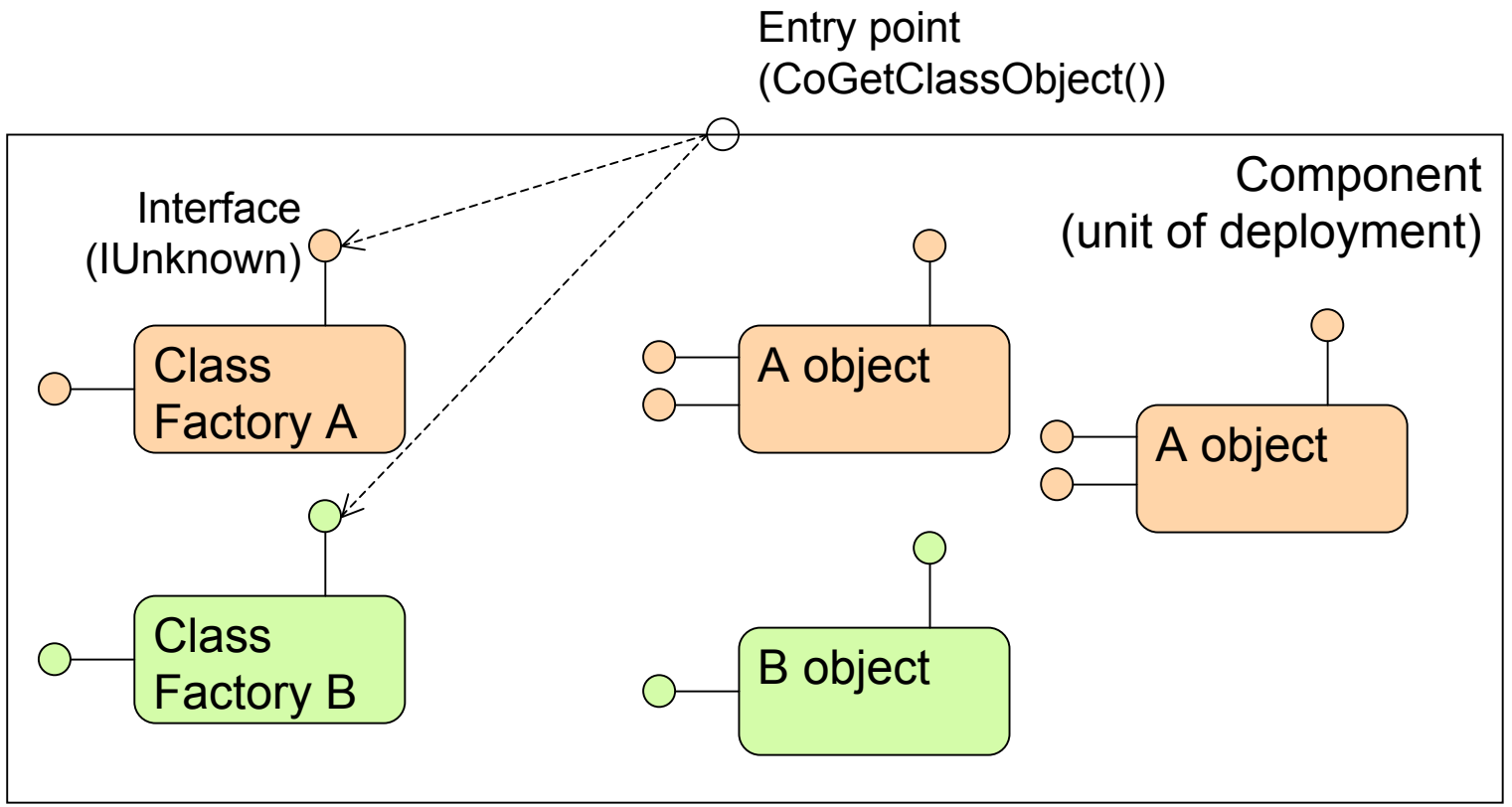
Components:

- are units of independent deployment;
- are units of third-party composition;
- have no persistent state.

Objects:

- are units of instantiation (each object has a unique identity);
- have state that can be persistent;
- encapsulate their state and behavior.

Components and Objects illustrated



Components are rather on the level of classes than of objects

Interfaces

- A component's interfaces define its access points.
  - These points let clients access the component's services.
  - Components may have multiple interfaces.
  - Each access point may provide a different service.

- Interface specifications have contractual nature.
  - Component and clients are developed in mutual ignorance.
  - The standardized contract forms ground for successful interaction.

Explicit Context Dependencies

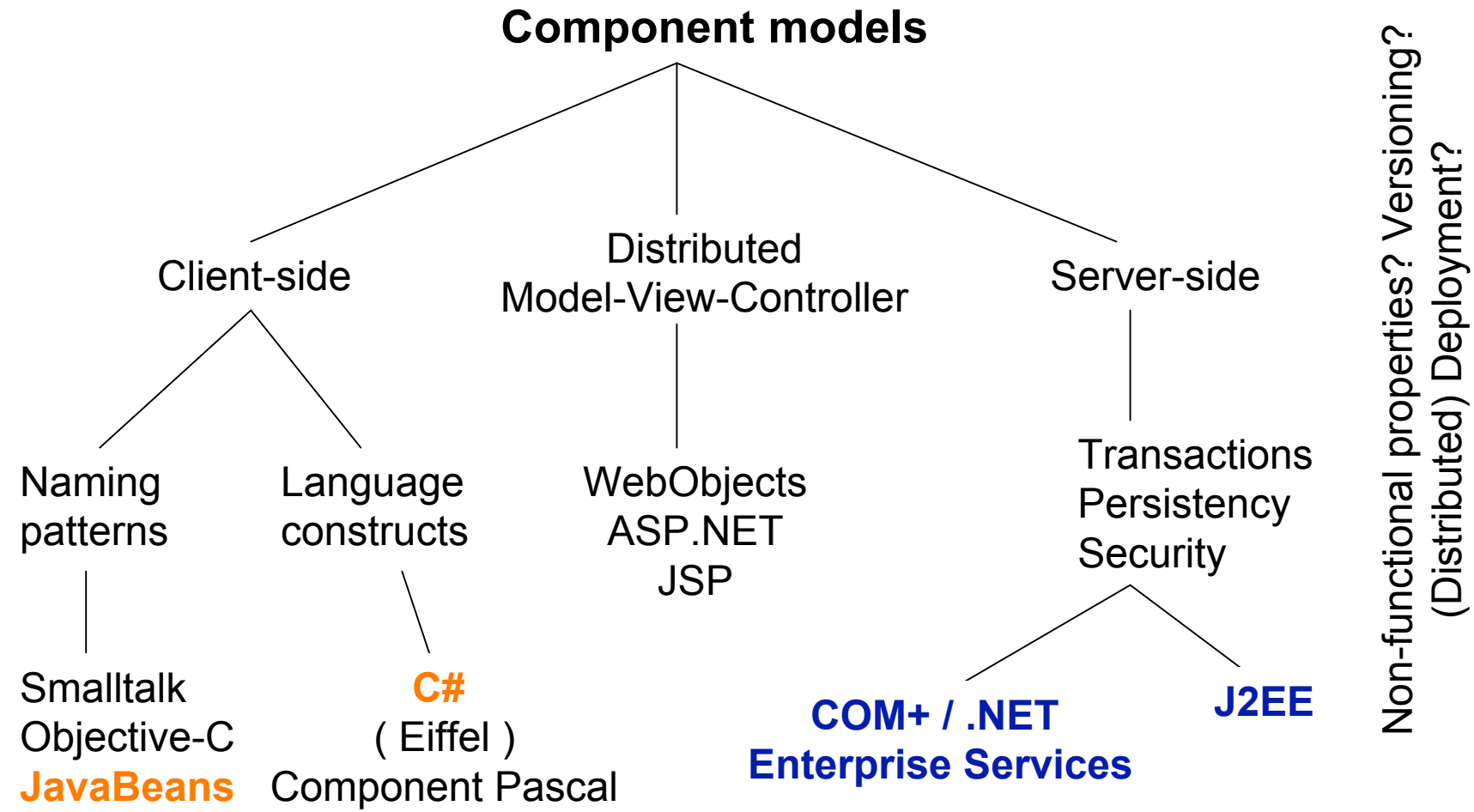
- Besides specifying provided interfaces, components are also required to specify their needs.
  - What does the deployment environment need to provide, so that the components can function (so-called context dependencies).
  - For example, a mail-merge component would specify that it needs a file system interface.
- Problems with today's components:
  - The list of required interfaces is not normally available.
  - Emphasis is usually just on provided interfaces.
- Non-functional component properties are not addressed
  - CPU/memory usage, timing behavior, fault-tolerance properties.

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.

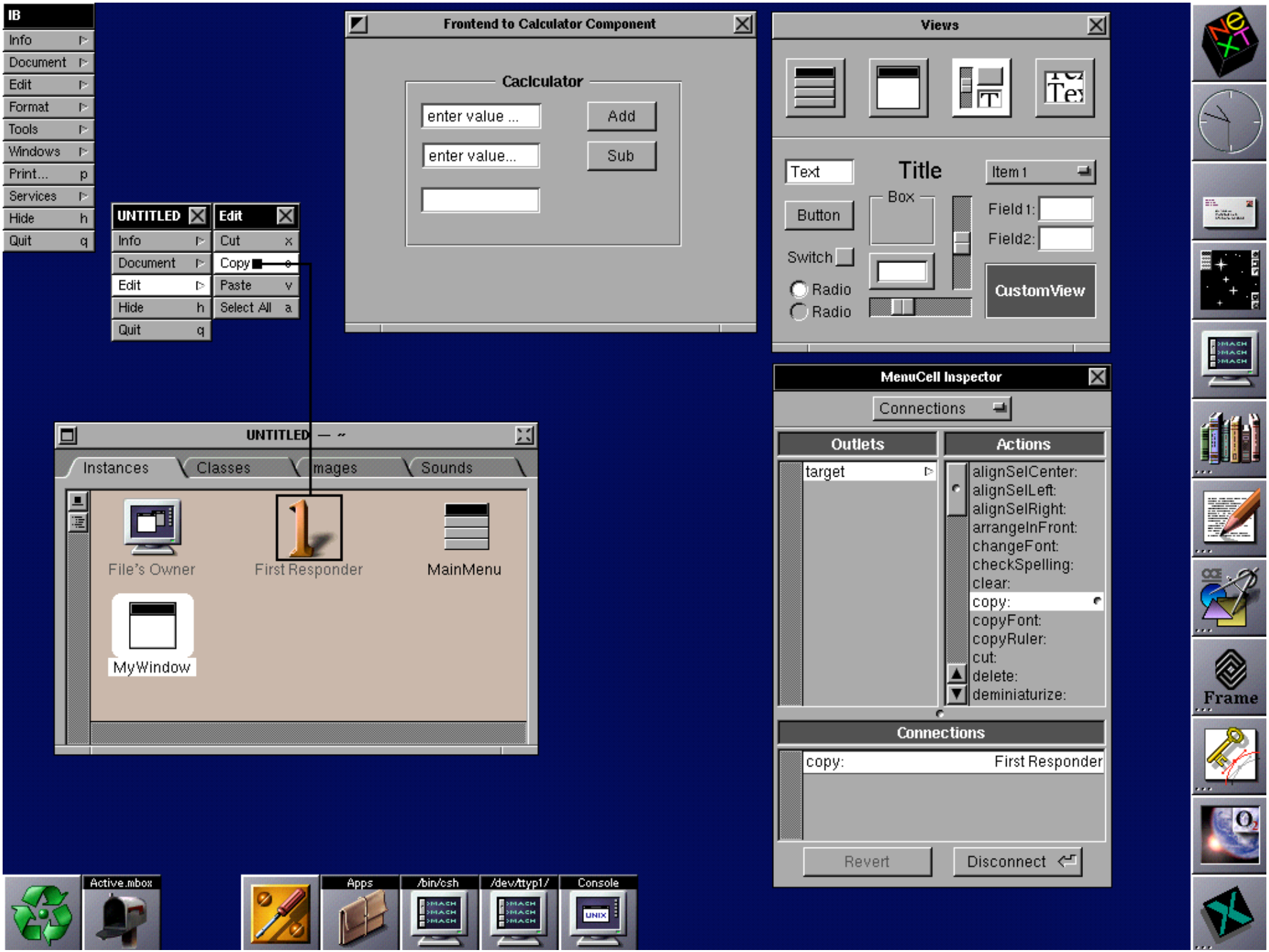
- Good software engineering practices strengthen the static description of systems as much as possible.
  - Dynamics can always be superimposed where needed.
  - Meta-programming and just-in-time compilation simplify this soft treatment of the boundary between static and dynamic.

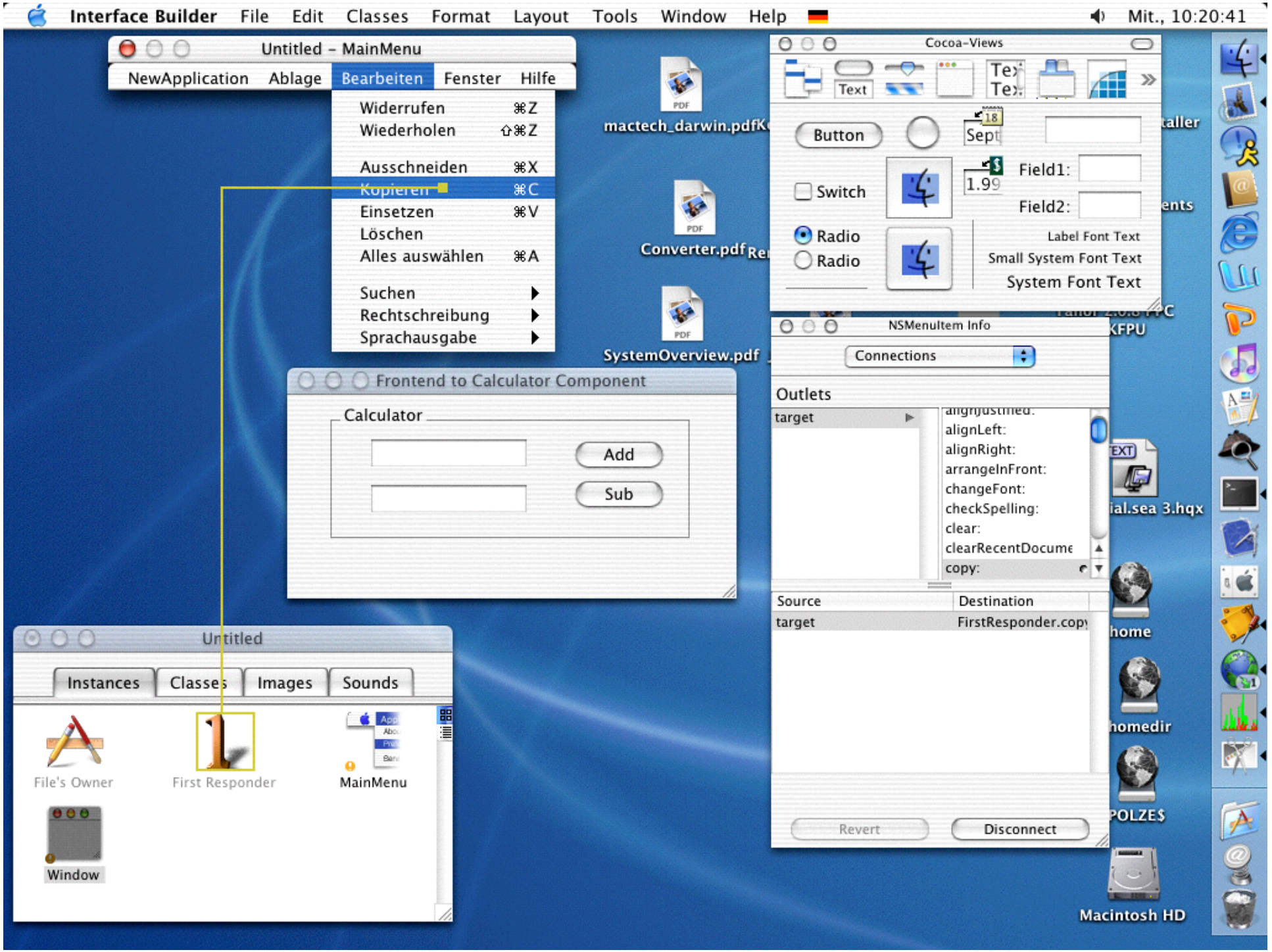
The Component Landscape



I Components at Work

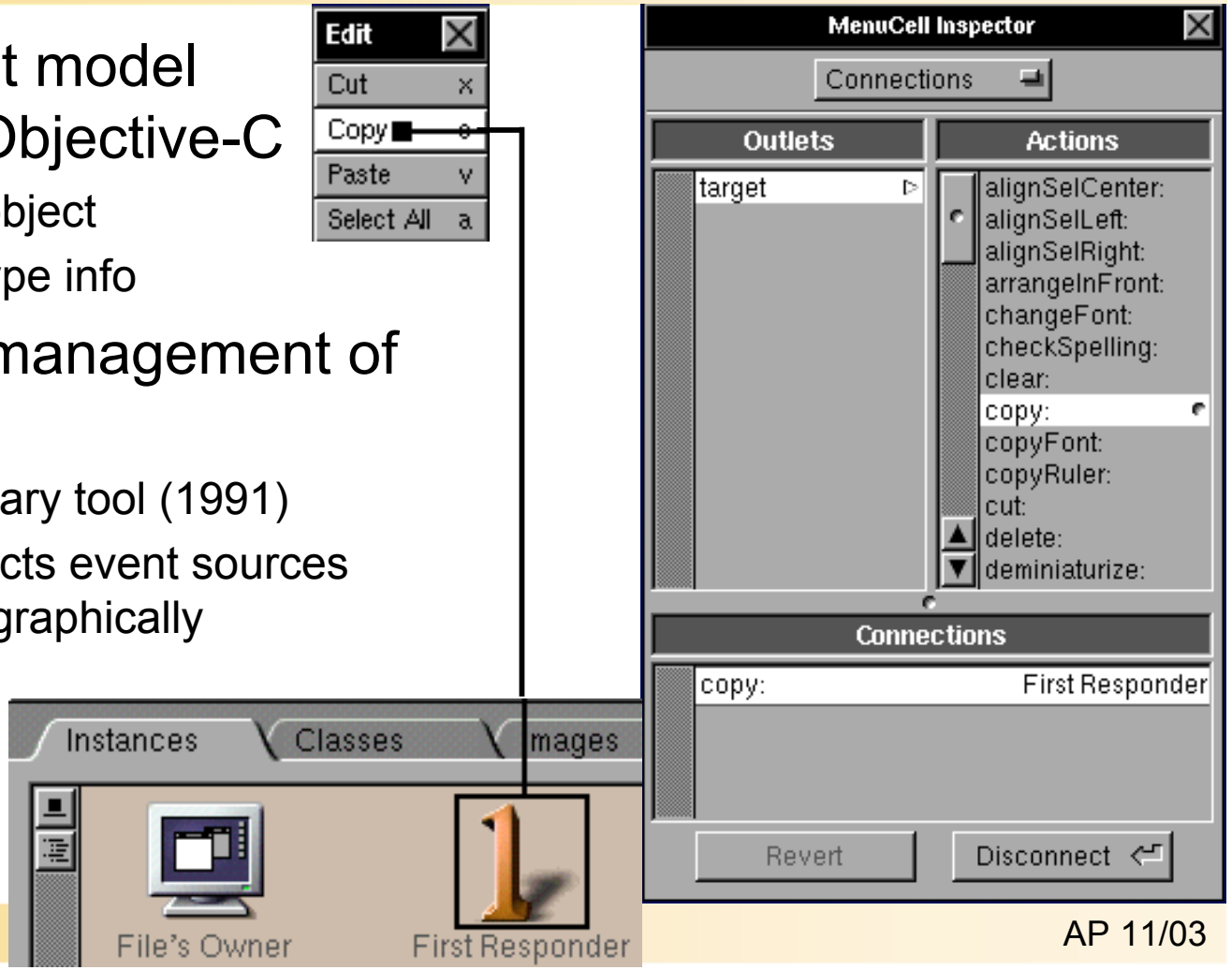
Setting the stage: NeXTSTEP and its tools


IB
Info
Document
Edit
Format
Tools
Windows
Print... p
Services
Hide h
Quit q
Frontend to Calculator Component
Caclculator
enter value ...
Add
enter value...
Sub
Views
Text
Title
Item 1
Box
Button
Field 1:
Field2:
Switch
Radio
Radio
CustomView
UNTITLED
Info
Document
Edit
Hide h
Quit q
Edit
Cut x
Copy c
Paste v
Select All a
UNTITLED — ~
Instances
Classes
Images
Sounds
File's Owner
First Responder
MainMenu
MyWindow
MenuCell Inspector
Connections
Outlets
target
Actions
alignSelCenter:
alignSelLeft:
alignSelRight:
arrangeInFront:
changeFont:
checkSpelling:
clear:
copy:
copyFont:
copyRuler:
cut:
delete:
deminiaturize:
Connections
copy: First Responder
Revert
Disconnect
Active.mbox
Apps
/bin/csh
/dev/ttyp1/
Console
Frame

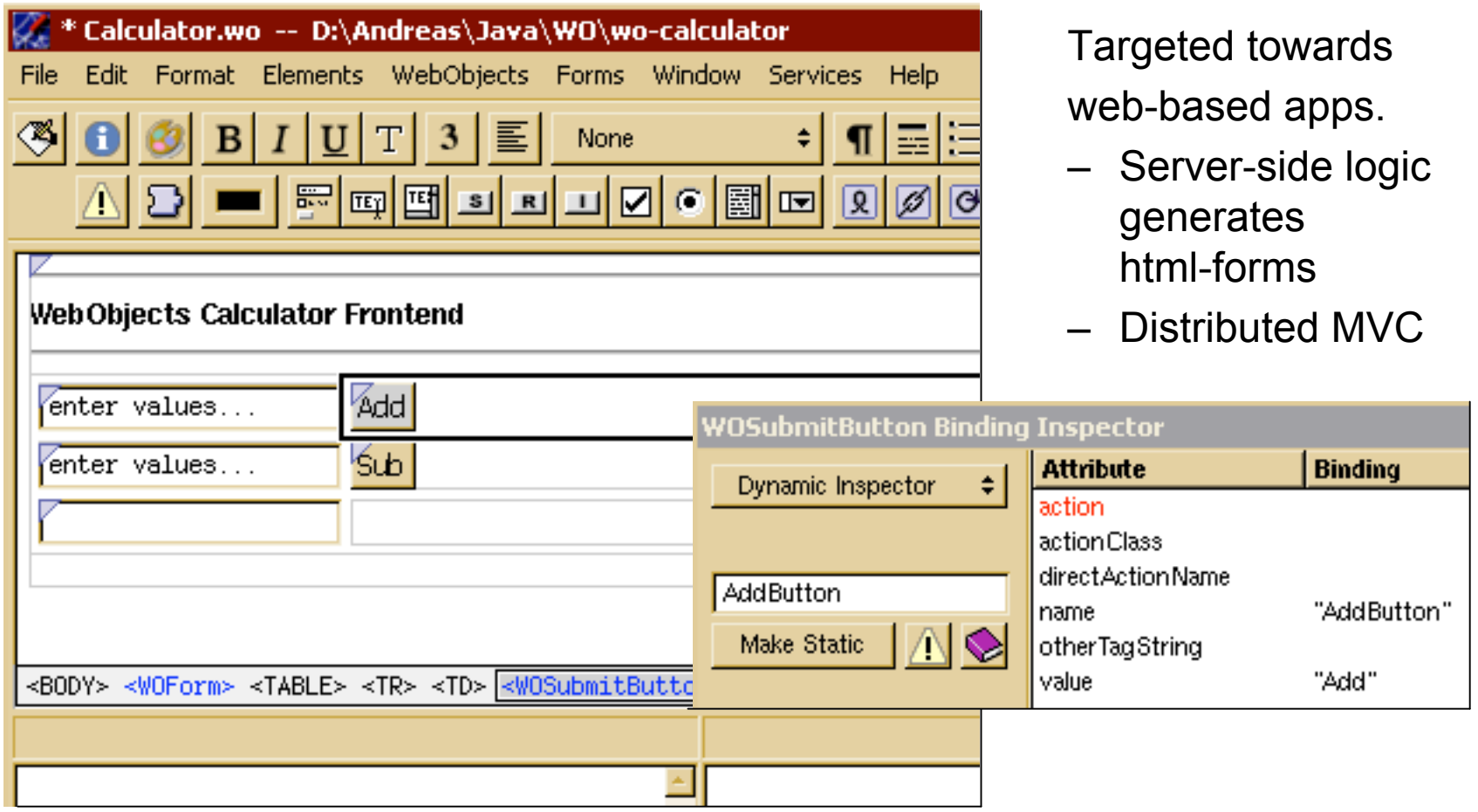

Interface Builder File Edit Classes Format Layout Tools Window Help
Mit., 10:20:41
Untitled – MainMenu
NewApplication Ablage Bearbeiten Fenster Hilfe
Widerrufen ⌘Z
Wiederholen ⇧⌘Z
Ausschneiden ⌘X
Kopieren ⌘C
Einsetzen ⌘V
Löschen
Alles auswählen ⌘A
Suchen
Rechtschreibung
Sprachausgabe
Cocoa-Views
Text
Button
Switch
Radio
Radio
Field1:
Field2:
1.99
Label Font Text
Small System Font Text
System Font Text
Frontend to Calculator Component
Calculator
Add
Sub
NSMenuItem Info
Connections
Outlets
target
alignLeft:
alignRight:
arrangeInFront:
changeFont:
checkSpelling:
clear:
clearRecentDocume
copy:
Source
Destination
target
FirstResponder.copy
Revert
Disconnect
Untitled
Instances
Classes
Images
Sounds
File's Owner
First Responder
MainMenu
Window
mactech_darwin.pdf
Converter.pdf
SystemOverview.pdf
home
homedir
POLZE$
Macintosh HD

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



WebObjectsBuilder on Windows 2000



Targeted towards web-based apps.

- Server-side logic generates html-forms
- Distributed MVC

II

Components at Work

The JavaBeans component model

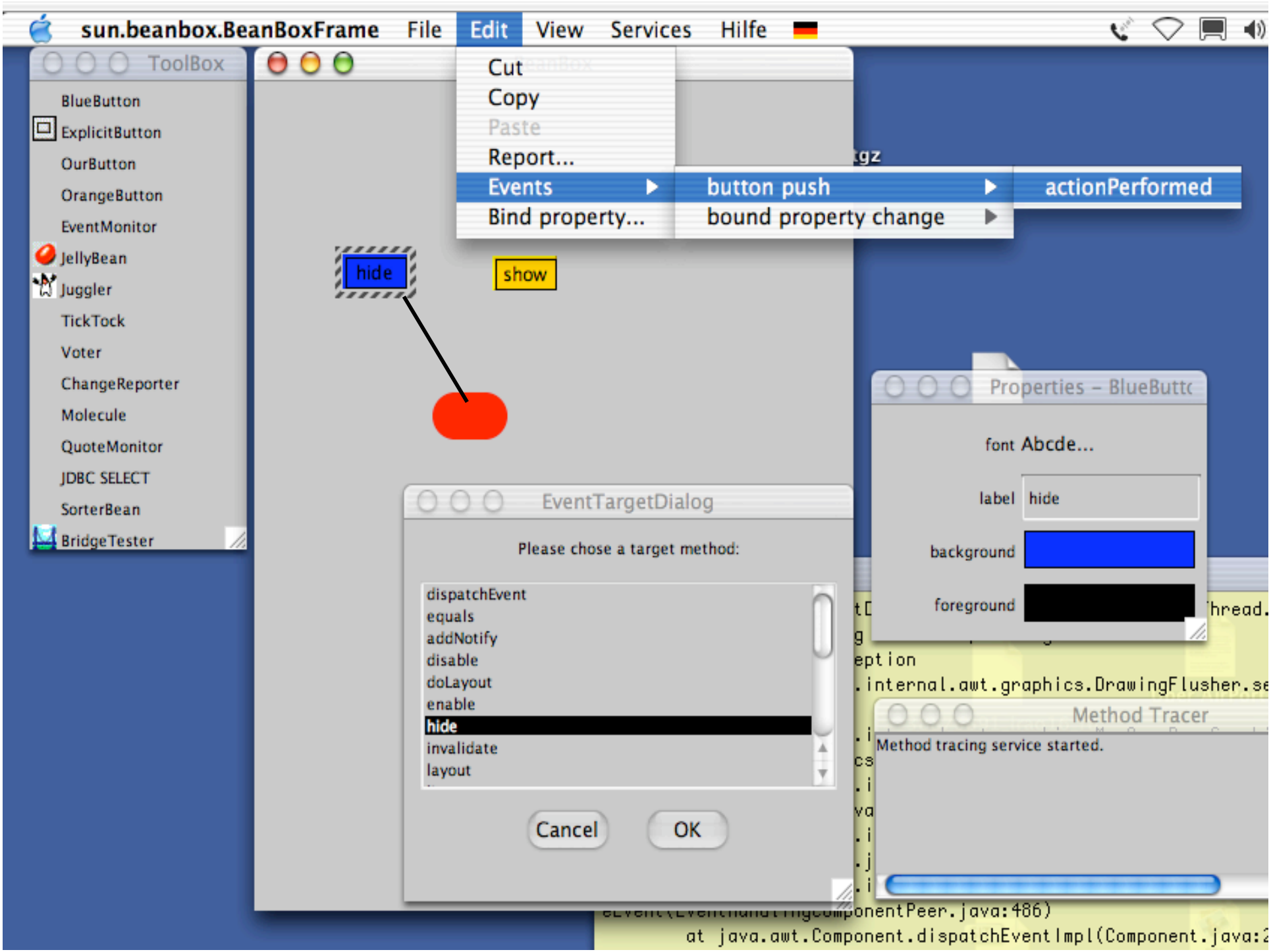
JavaBeans - based on **Naming Patterns**

- 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.


sun.beanbox.BeanBoxFrame
File
Edit
View
Services
Hilfe
ToolBox
BlueButton
ExplicitButton
OurButton
OrangeButton
EventMonitor
JellyBean
Juggler
TickTock
Voter
ChangeReporter
Molecule
QuoteMonitor
JDBC SELECT
SorterBean
BridgeTester
Cut
Copy
Paste
Report...
Events
Bind property...
button push
bound property change
actionPerformed
hide
show
Properties – BlueButtc
font Abcde...
label
hide
background
foreground
EventTargetDialog
Please chose a target method:
dispatchEvent
equals
addNotify
disable
doLayout
enable
hide
invalidate
layout
Cancel
OK
Method Tracer
Method tracing service started.
at java.awt.Component.dispatchEventImpl(Component.java:2

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)

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

- Public methods describe the Bean's behavior.
  - Excluding methods that get/set property values
  - Excluding methods that register/remove event listeners

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.
- Inter-Bean dependencies:
  - **Design-Time-Only** and **Depends-On** tags. (JavaBeans 1.01)

- 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;

III

Components in C#

What defines a component?

- What defines a component?
  - Properties, methods, events
  - Design-time and runtime information
  - Integrated help and documentation

Language constructs for
- Indexers
- Properties
- events

- C# has first class component support
  - Not naming patterns, adapters, etc.
  - Not external files
  - Versionable (side-by-side execution, global assembly cache)

- Easy to build and consume

Extensible component metadata
- attributes

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

```
[WebMethod]
  void GetCustomers() { … }
```

COM Support

- .NET Framework provides COM support
  - TLBIMP imports existing COM classes
  - TLBEXP exports .NET types
- C# exposes features of the framework most naturally
  - Easy to consume COM+ services
  - MTS plays the role of an application server
- .NET components can be hosted server-side
  - IIS is .NET's application server
  - Limited set of features - but easily interoperable with COM+

IV

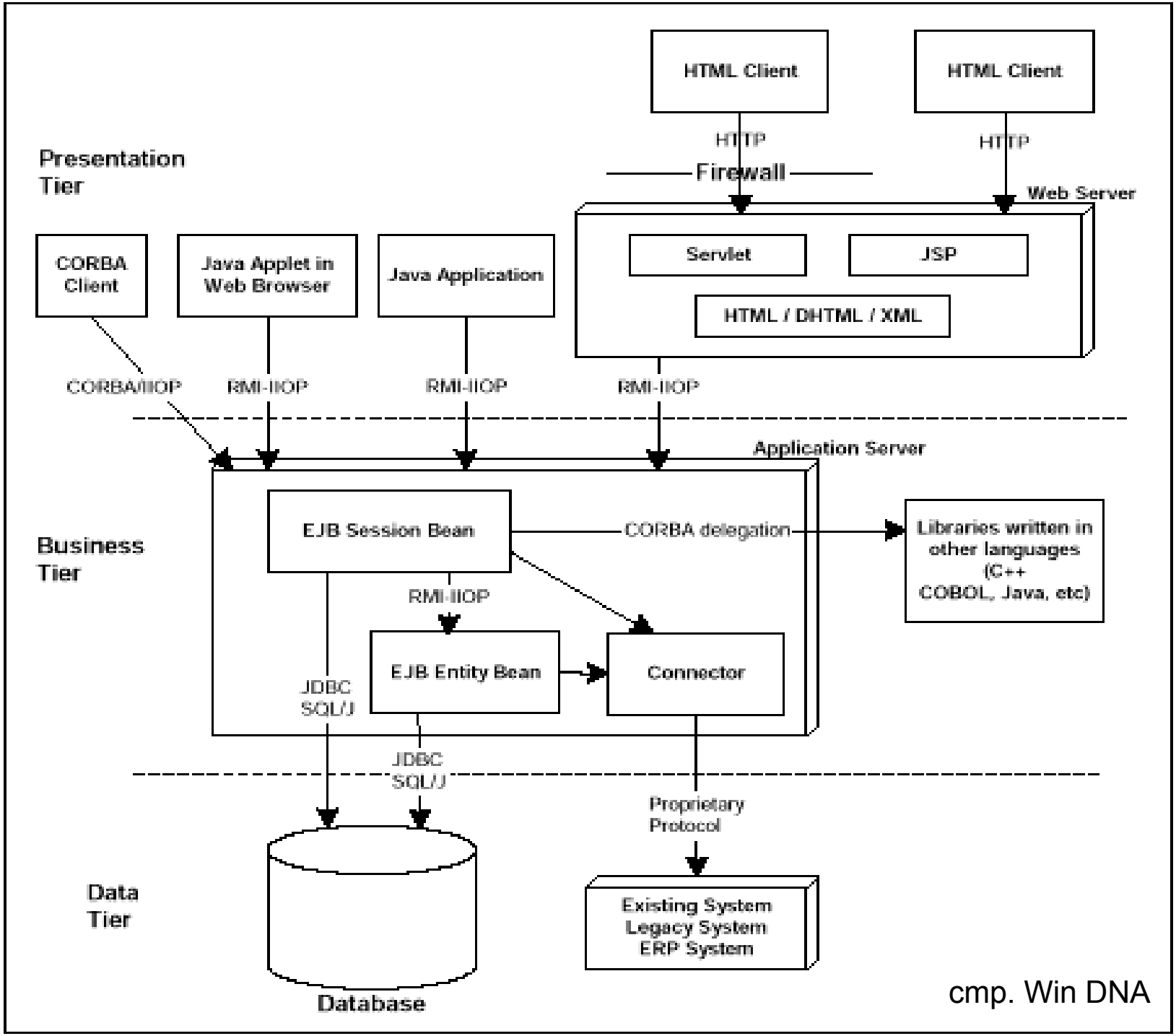
Enterprise Java Beans

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.

EJB Object Model



cmp. Win DNA

Enterprise JavaBeans Component Model

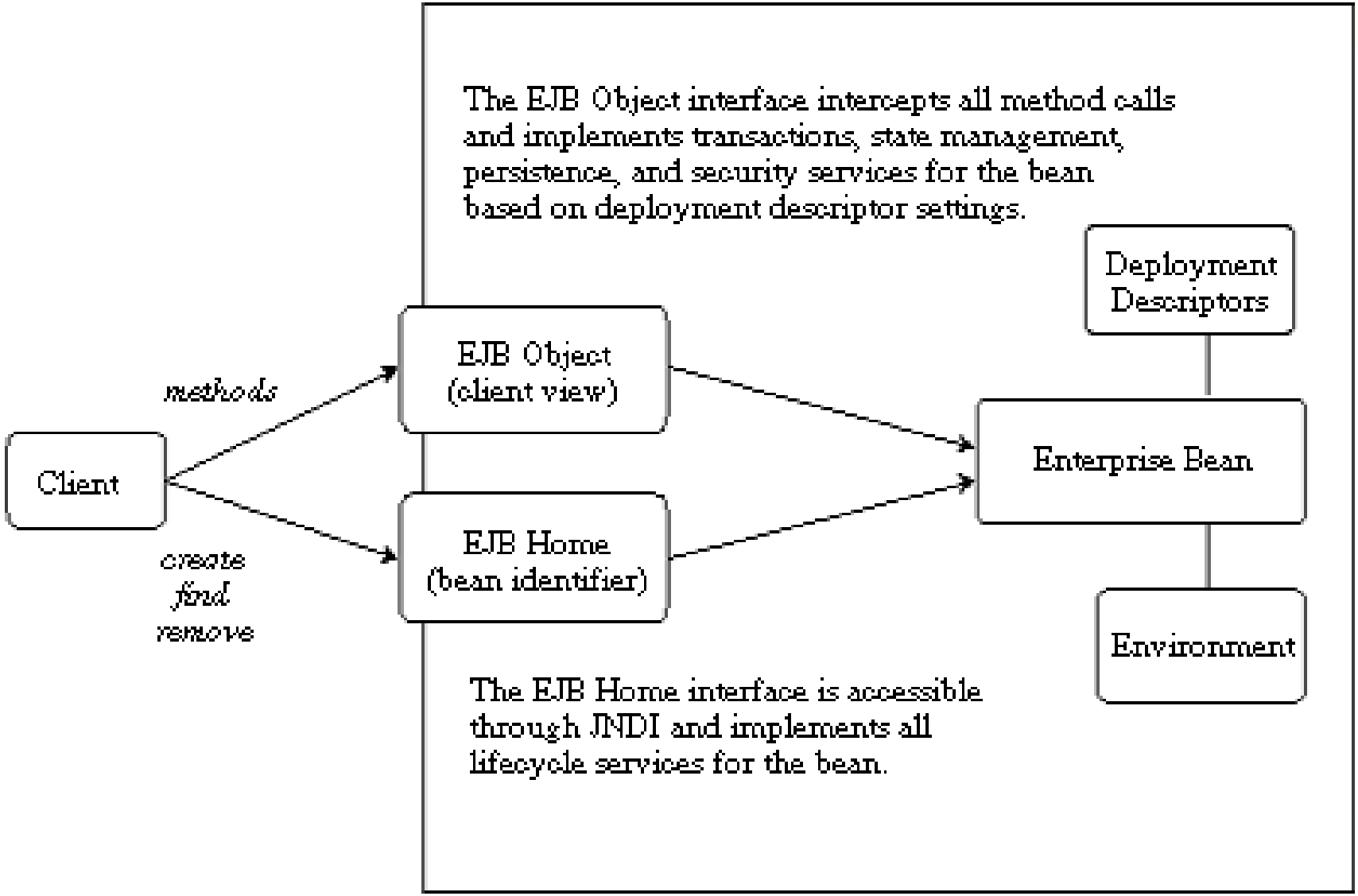
- Server components
  - are reusable, prepackaged pieces of application functionality that are designed to run in an application server.
  - They can be combined with other components to create customized application systems.
- Enterprise JavaBeans components (enterprise beans)
  - cannot be manipulated 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.

EJB - Implicit Services

EJB container performs services on behalf of the enterprise beans

- Lifecycle
  - Process allocation, thread management, object activation, object destruction.
- State Management
  - Save or restore conversational object state between method calls.
- Security
  - Authenticate users and check authorization levels.
- Transactions
  - The EJB container can automatically manage the start, enrollment, commitment, and rollback of transactions on behalf of the enterprise bean.
- Persistence
  - The EJB container can automatically manage persistent data on behalf of the enterprise bean.

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.

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.

The Component Object Model (COM+)

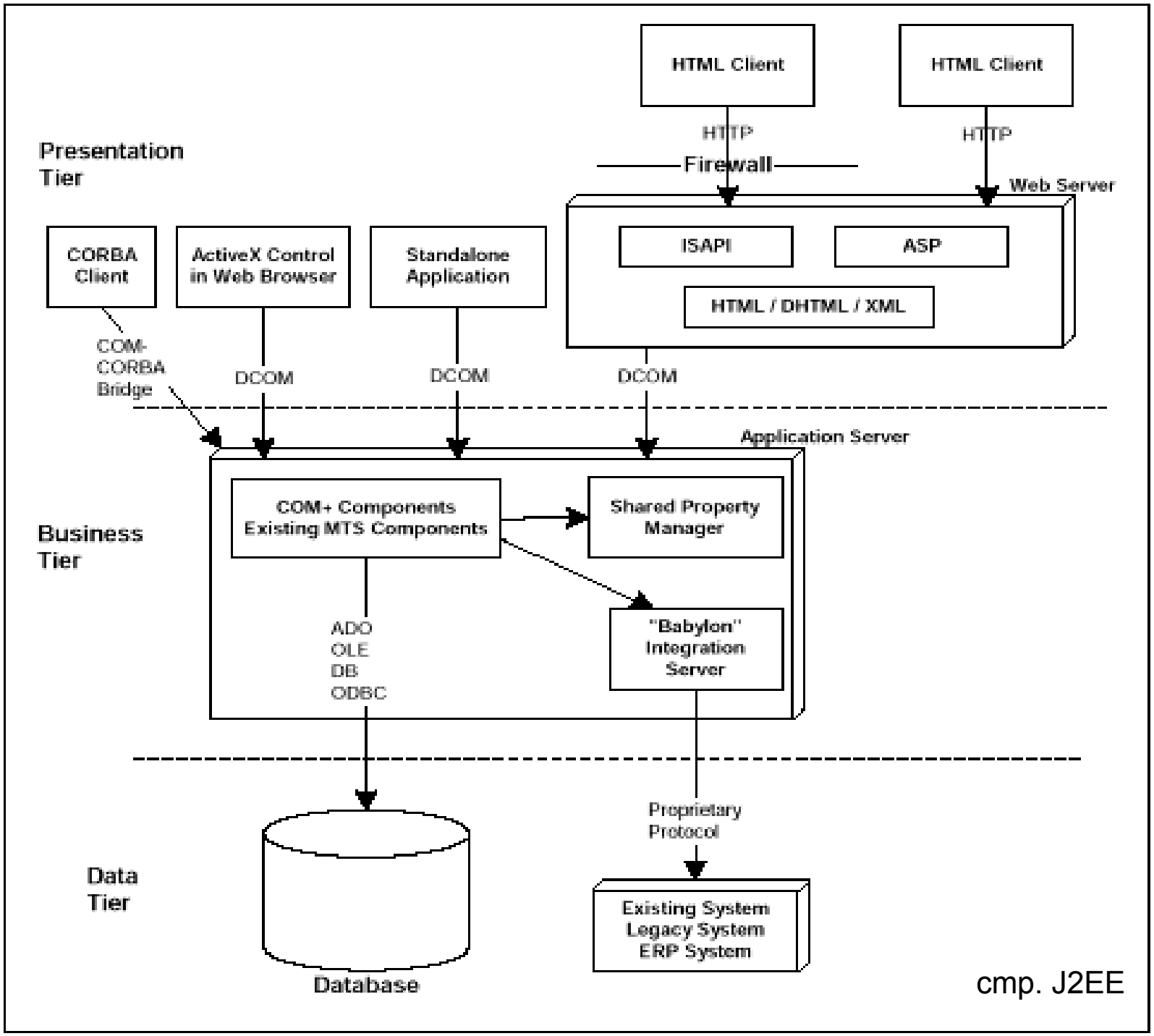
also known as

.NET Enterprise Services

Microsoft Transaction Server (MTS)

- MTS provides container system for COM server comp.
  - providing transactional and security services similar to those provided in Enterprise JavaBeans servers.
- First Component Transaction Monitor in the market
  - Server-side component model (COM)
  - Distributed component service based on DCOM
- COM+ / .NET Enterprise Services
  - the next generation of MTS
  - provides additional capabilities, such as dynamic load-balancing and queued request-processing.

Microsoft Windows DNA Object Model



cmp. J2EE

.NET Enterprise Servers

- Actually released before .NET; based on Windows DNA
- BizTalk Server 2000:
  - XML Messaging Engine, Orchestration Engine
- Application Center 2000:
  - Manages complexity of Windows DNA application deployment
- Host Integration Server (HIS) 2000:
  - COM Transaction Integrator - interface to IBM TM
  - OLE for DB2 provider
  - MSMQ-MQSeries bridge
- SQL Server 2000:
  - XML support, primary DBMS product

- Exchange Server 2000, Mobile Information Server 2001, Internet Security and Acceleration Server (ISA) 2000

COM – The idea

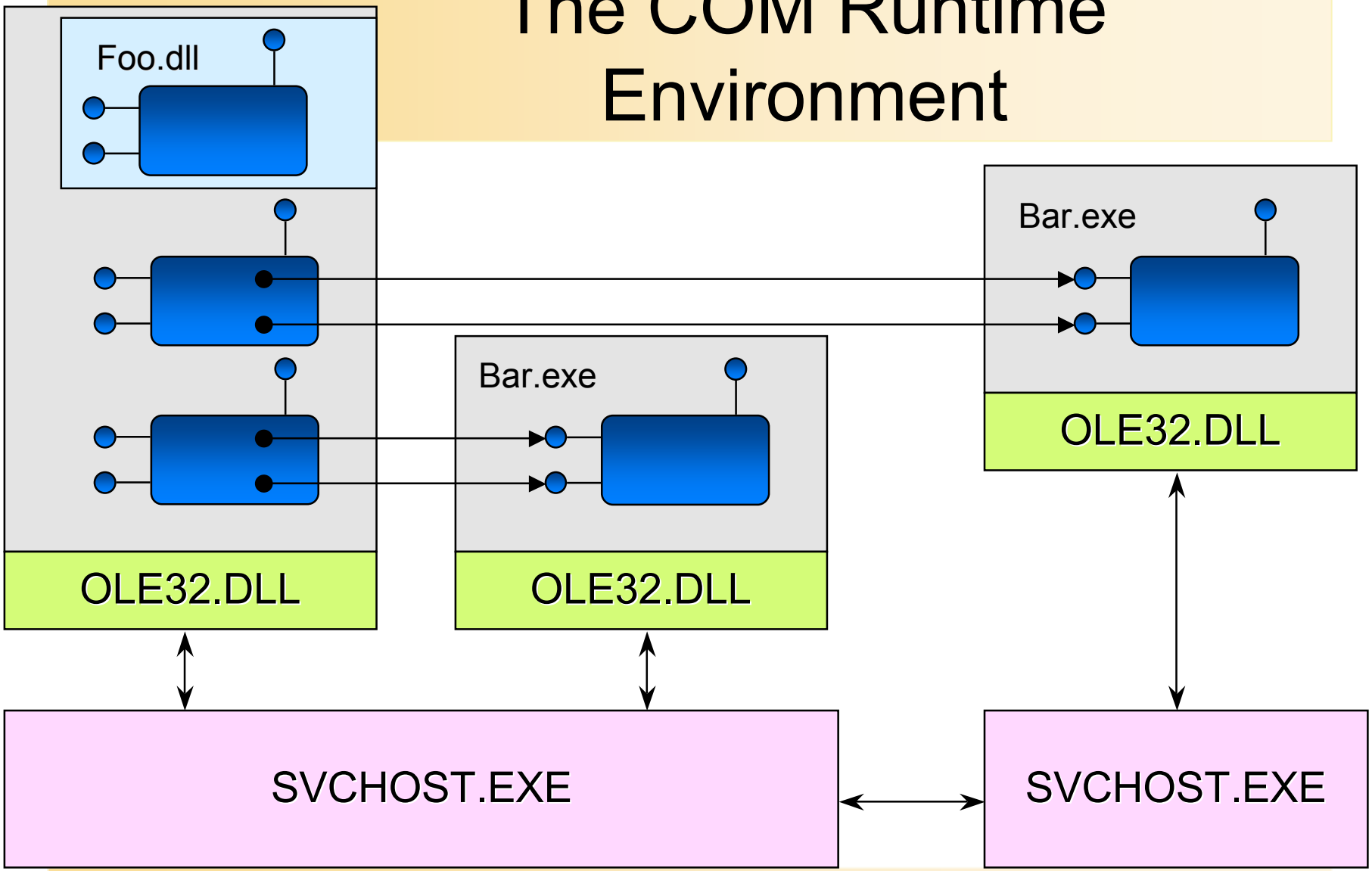
Three fundamental ideas:

- 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

- 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

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
  - **MTS may intercept SCM's operation**
- 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)
  - DllGetClassObject() is entry point into a COM component

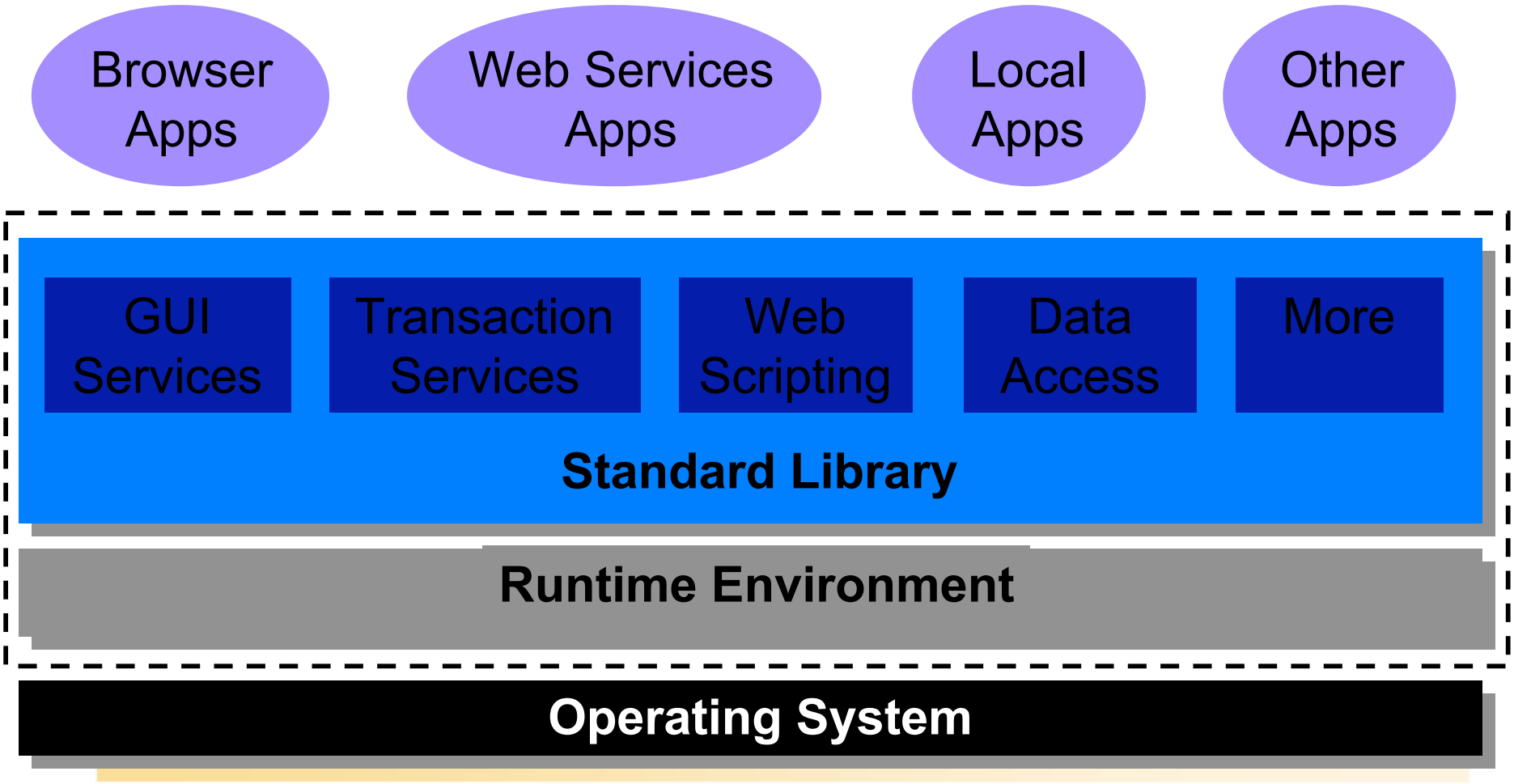
VI

Comparison
J2EE versus .NET

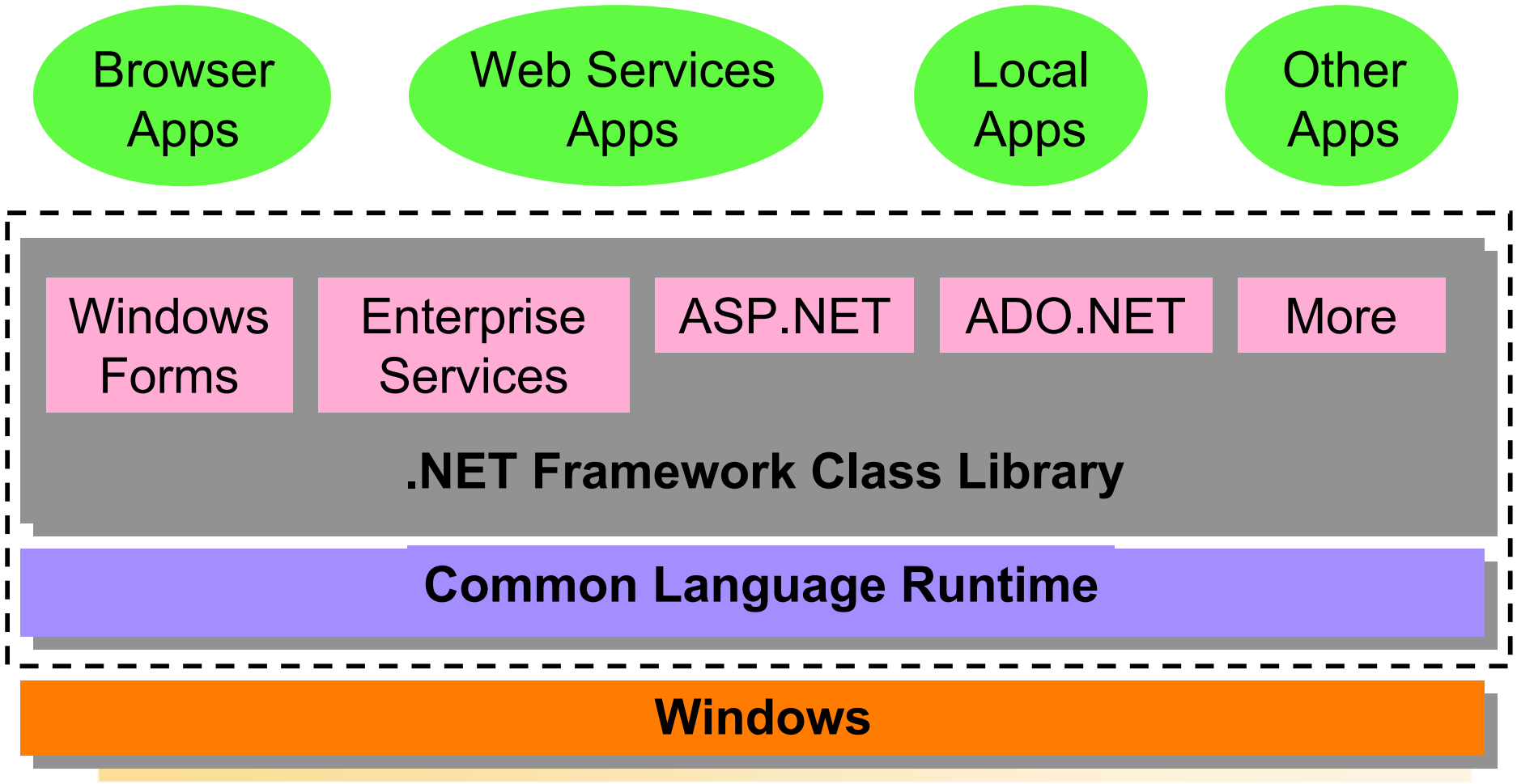
Areas for Comparison: Application Platforms

- .NET
  - The .NET Framework,
  - IIS - the application server, MTS - the component transaction monitor
- Java
  - Java application servers
  - Products include:
    - IBM WebSphere Application Server
    - BEA WebLogic Application Server
    - Sun ONE Application Server
    - Oracle Application Server
    - Many others

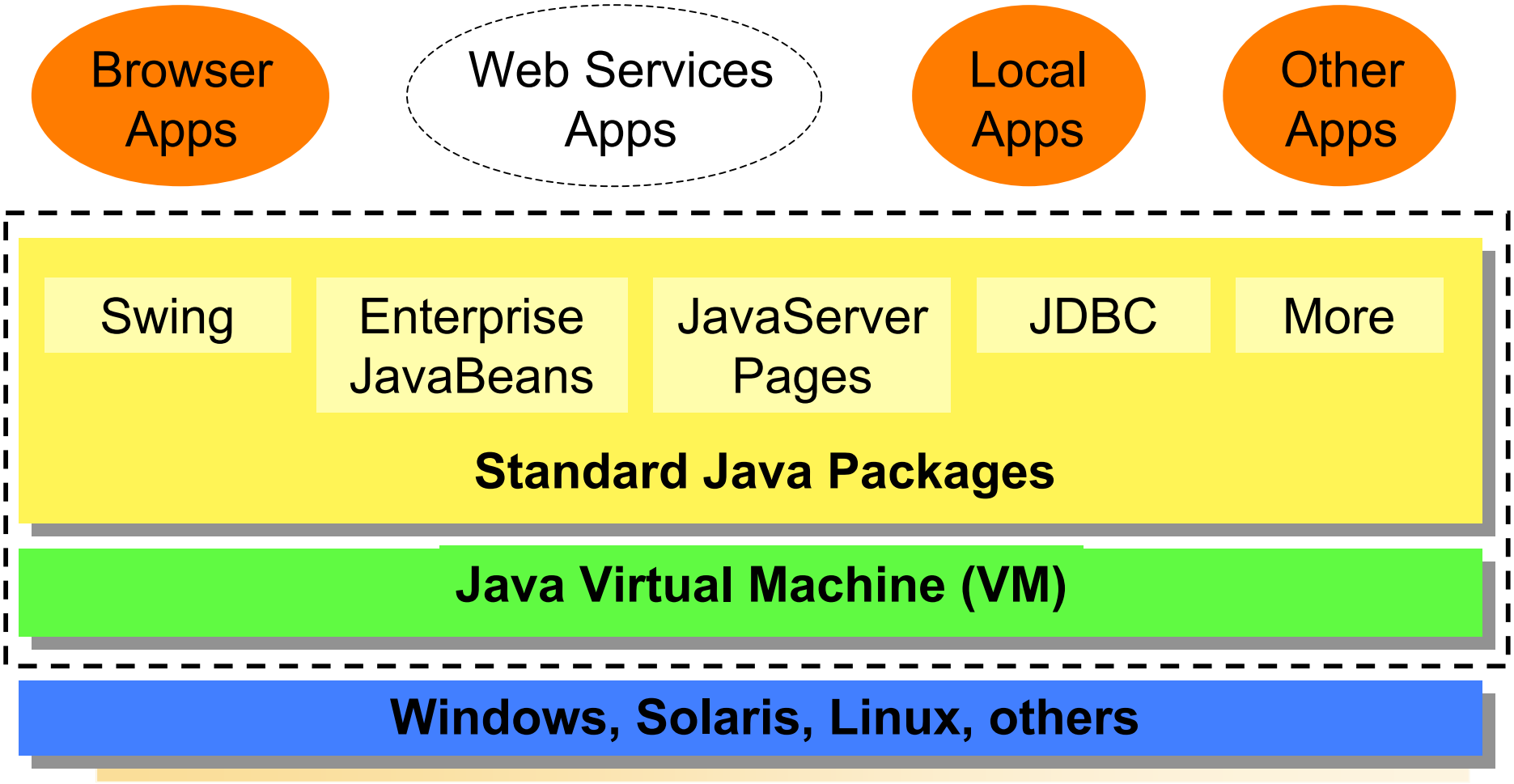
Application Platforms Today



Illustrating The .NET Framework



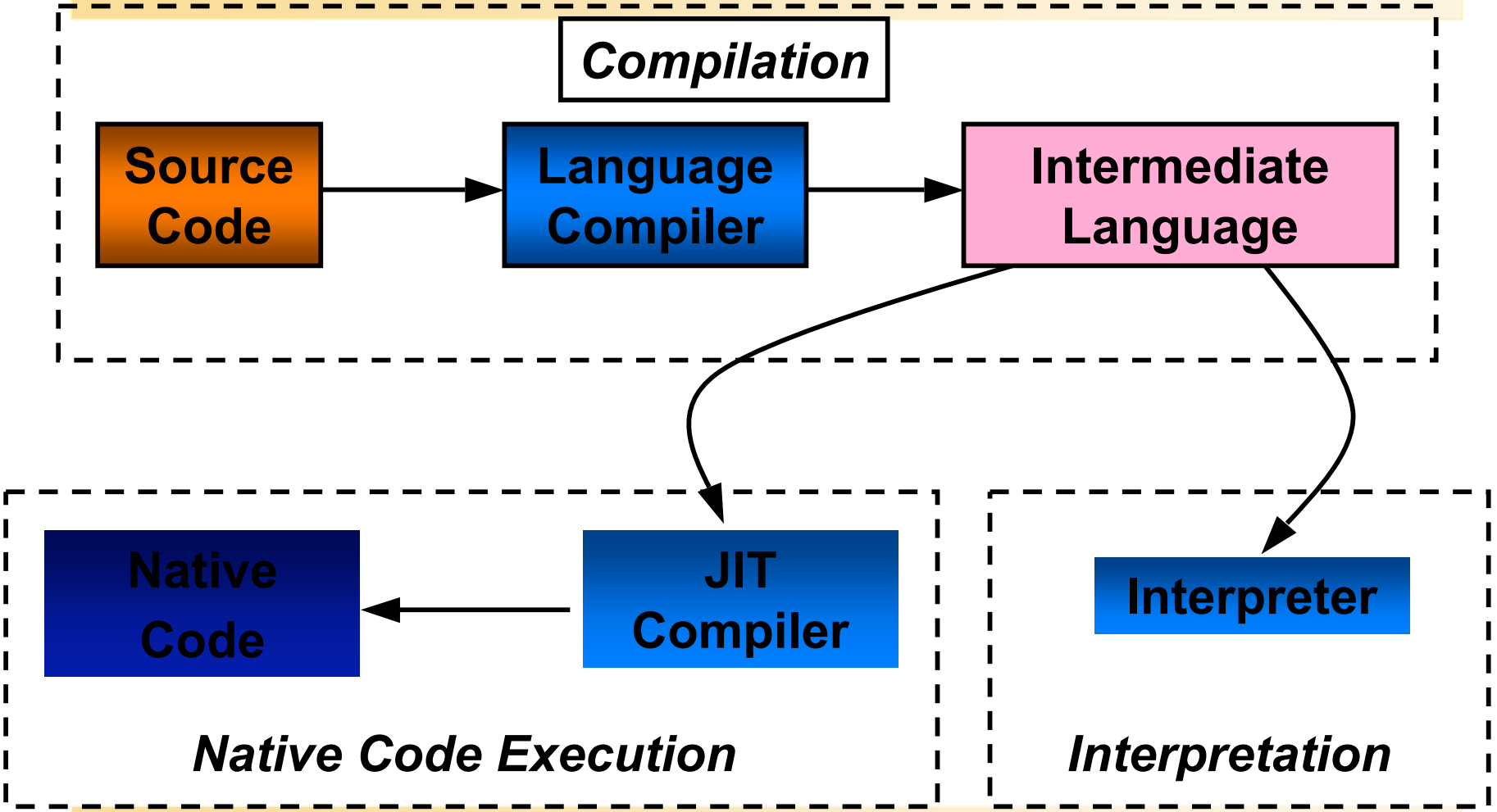
Illustrating The Java Environment



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*	

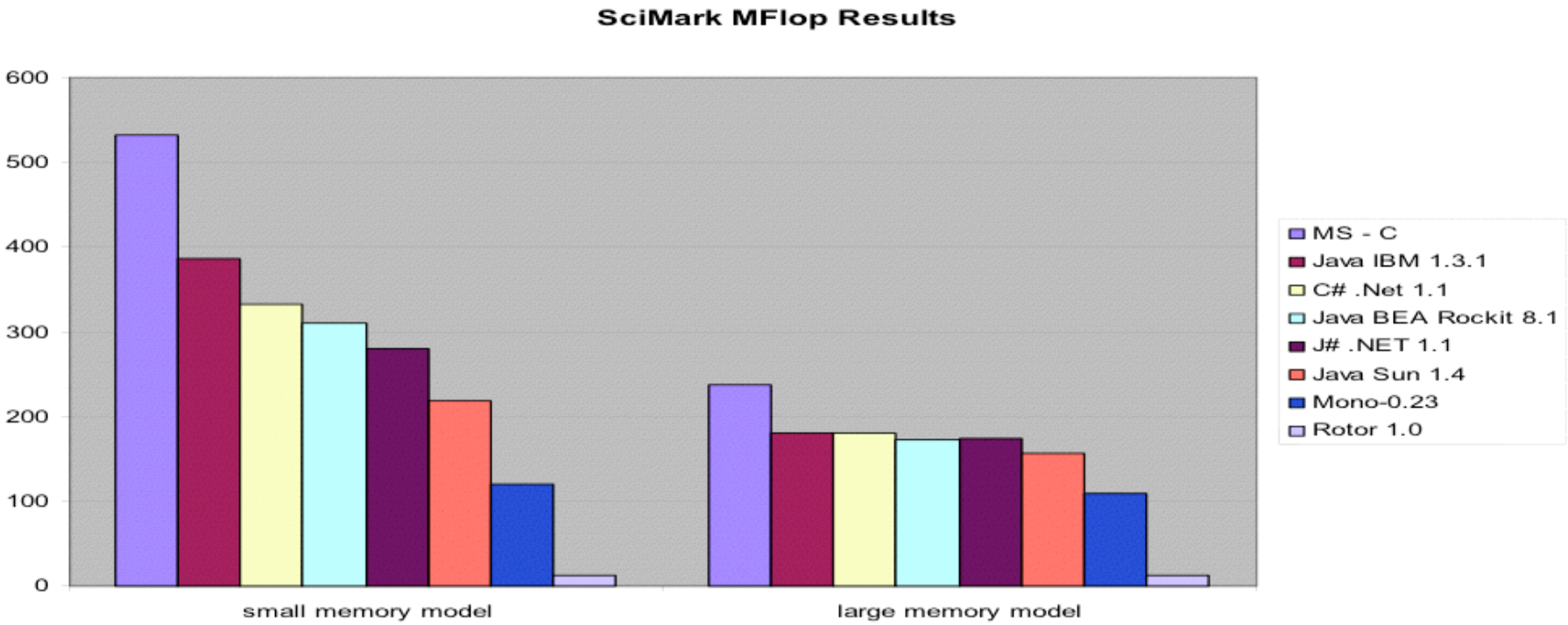
Runtime Environments Today



Areas for Comparison: Runtime Environments

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

Performance Comparison



from Vogels, W., „HPC.NET - are CLI-based Virtual Machines Suitable for High-Performance Computing?“, in Proc. of SuperComputing 2003, Phoenix, AZ, Nov. 2003.

Summarizing the Technologies (1)

	.NET	***Java***
Application Server	.NET Framework IIS, MTS (COM+)	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
  - Developer productivity
- Operating system support
- Cost

VII

Predictability

Non-functional properties:

Aspect-Oriented Programming with C# and .NET

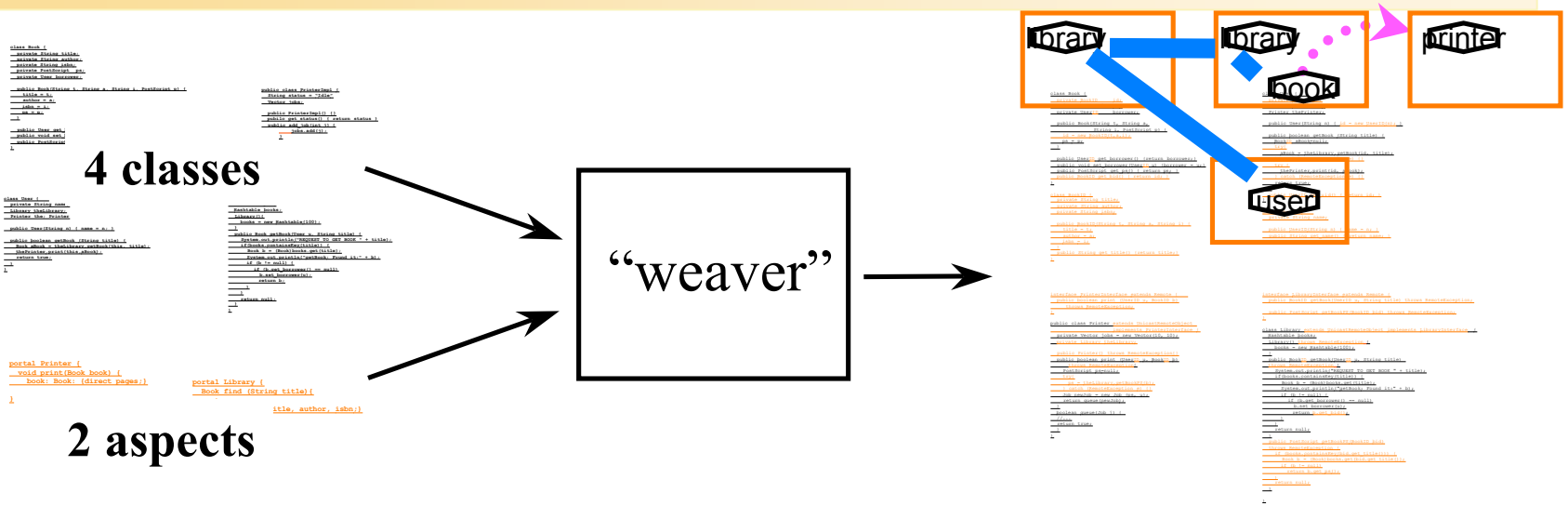
Replication based on Attributes

- Specification of a component's non-functional properties at design time
- A tool may generate code to automatically create replicated objects
- Component behavior described by user-defined attributes

```
namespace CalculatorClass {
    using System; using proxy;

    [TolerateCrashFaults(4)]
    public class Calculator  {
            ...
            public double add
                (double x, double y) {
                    return x + y;
            }
    }
}
```

AOP - Cross Cutting Concerns

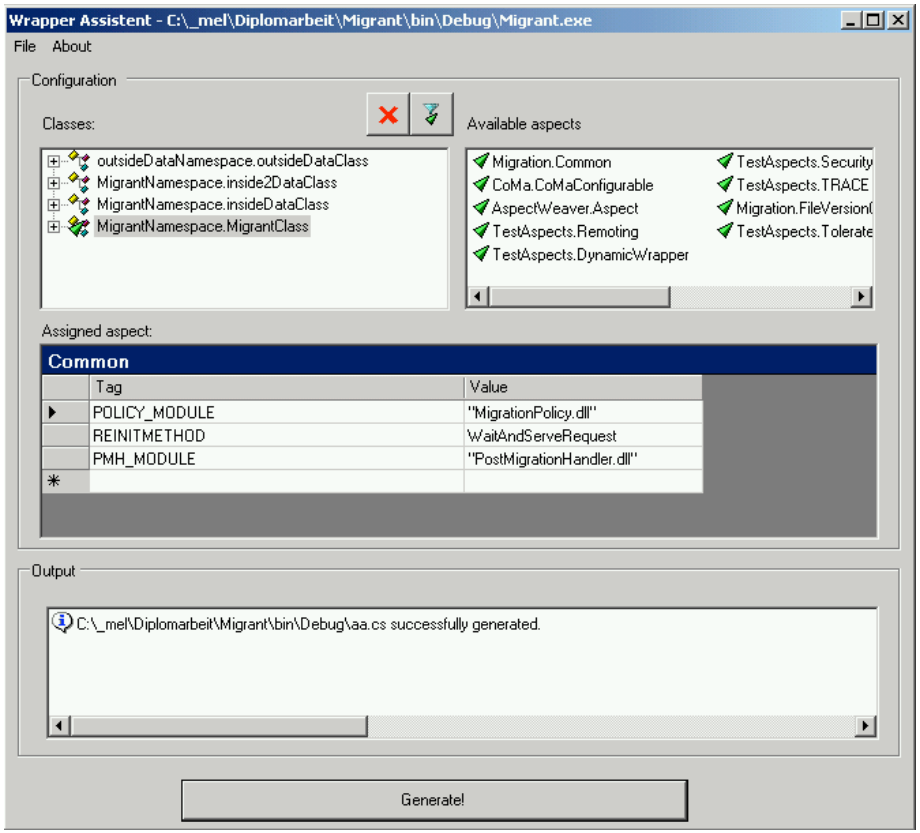


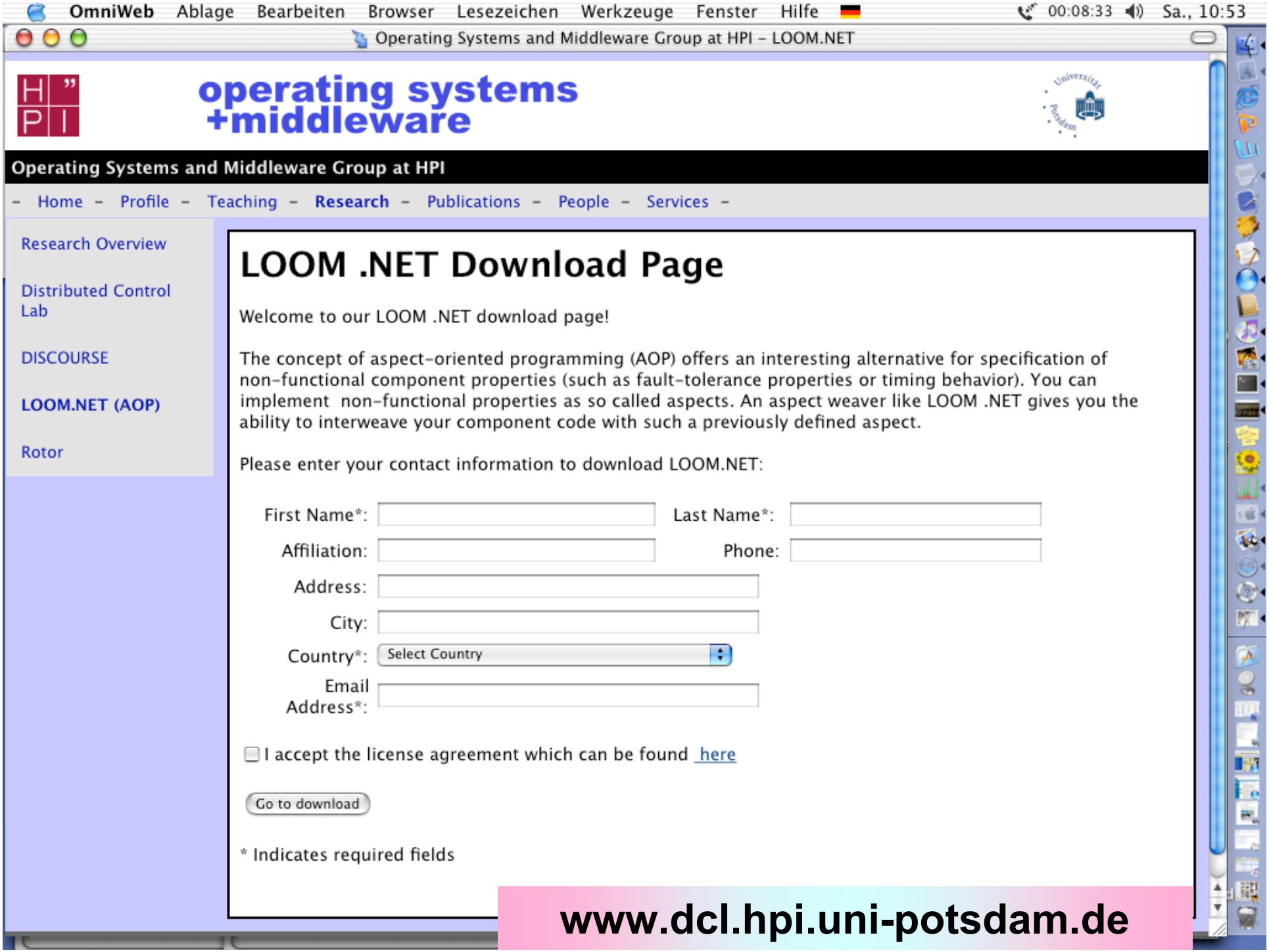
- a “language processor” that
  - accepts two kinds of code as input;
  - produces “woven” output code, or
  - directly implements the computation

(more at www.parc.xerox.com/csl/projects/aop/)

LOOM.NET - AOP Framework for .NET

- An aspect is defined as a set of templates for classes, methods, ...
- Weaver generates new code and builds the application
- Weaving can be restricted to chosen entities (e.g. methods)
- Aspects can be parameterized




OmniWeb Ablage Bearbeiten Browser Lesezeichen Werkzeuge Fenster Hilfe
00:08:33 Sa., 10:53
Operating Systems and Middleware Group at HPI – LOOM.NET
operating systems +middleware
Operating Systems and Middleware Group at HPI
– Home – Profile – Teaching – Research – Publications – People – Services –
Research Overview
Distributed Control Lab
DISCOURSE
LOOM.NET (AOP)
Rotor
LOOM .NET Download Page
Welcome to our LOOM .NET download page!
The concept of aspect-oriented programming (AOP) offers an interesting alternative for specification of non-functional component properties (such as fault-tolerance properties or timing behavior). You can implement non-functional properties as so called aspects. An aspect weaver like LOOM .NET gives you the ability to interweave your component code with such a previously defined aspect.
Please enter your contact information to download LOOM.NET:
First Name*:
Last Name*:
Affiliation:
Phone:
Address:
City:
Country*:
Select Country
Email Address*:
I accept the license agreement which can be found here
Go to download
* Indicates required fields
www.dcl.hpi.uni-potsdam.de

VIII

Conclusions

Conclusions

- Development world has bifurcated
  - Microsoft .NET
  - The Java environment
- Both have similar architectures
  - Both will (eventually) be interoperable
- Both will survive
  - Which is a good thing
- .NET will dominate in the Windows environment

Conclusions (contd.)

Evolution, not revolution:

- Microsoft .NET
  - C# supports client-side component programming extremely well (packaging, versioning, distributed deployment - in conjunction with group policies)
  - C# can easily consume COM+/MTS Enterprise Services
  - Watch out for next generation of MS AppServers (i.e.; BizTalk 2004)
- The Java environment
  - Client-side JavaBeans have some weaknesses compared to .NET (versioning, based on naming patterns)
  - J2EE is a mature technology
- Can you see the CORBA Component Model?

References

- Simon Guest, “Microsoft .NET and J2EE Interoperability Toolkit”, MS Press, ISBN 0-7356-1922-0, 2004.

- Richard Monson-Haefel, “Enterprise JavaBeans (2nd. Ed.)”, O’Reilly & Associates, ISBN 1-56592-869-5, 2000.

- David Chappell, “Understanding .NET”, Addison-Wesley, ISBN 0201741628, 2002.