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
IBM
closed proprietary

Software
Microsoft
closed proprietary

Middleware
???
open standards, WebServices

PC1 → PC2 → PC3

Cluster
SuperDome

SETI@home
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

Entry point (CoGetClassObject())

Component (unit of deployment)

Interface (IUnknown)

Class Factory A

Class Factory B

A object

B object

A object
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

Component models

Client-side
- Naming patterns
  - Smalltalk
  - Objective-C
  - JavaBeans
- Language constructs
  - C# (Eiffel)
  - Component Pascal

Distributed Model-View-Controller
- WebObjects
- ASP.NET
- JSP

Server-side
- Transactions
- Persistency
- Security
- Non-functional properties? Versioning?
- (Distributed) Deployment?
- COM+ / .NET
- Enterprise Services
- J2EE
I

Components at Work

Setting the stage:
NeXTSTEP and its tools
Déjà vu
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
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
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.
Properties

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

- **Getter:**
  - \texttt{public T getP()}
- **Boolean getter:**
  - \texttt{public boolean isP()}
- **Setter:**
  - \texttt{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
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;
Components in C#
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 (side-by-side execution, global assembly cache)

Easy to build and consume

Language constructs for
- Indexers
- Properties
- Events

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

```csharp
[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+
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.
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.
The EJB Object interface intercepts all method calls and implements transactions, state management, persistence, and security services for the bean based on deployment descriptor settings.

The EJB Home interface is accessible through JNDI and implements all lifecycle services for the bean.
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.
.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

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
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)
  – DIIGetClassObject() is entry point into a COM component
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

- Browser Apps
- Web Services Apps
- Local Apps
- Other Apps

Standard Library
- GUI Services
- Transaction Services
- Web Scripting
- Data Access
- More

Runtime Environment

Operating System
Illustrating The Java Environment

Standard Java Packages

Swing  Enterprise JavaBeans  JavaServer Pages  JDBC  More

Java Virtual Machine (VM)

Windows, Solaris, Linux, others
# Application Platforms: Some History

<table>
<thead>
<tr>
<th>Year</th>
<th>Microsoft</th>
<th>Java</th>
</tr>
</thead>
</table>
| 1996 | Windows DNA  
- MTS (now COM+)  
- ASP  
- ADO | Java  
- Java language  
- Java VM  
- J2SE |
| 1998 | .NET Framework  
- CLR  
- C#, VB.NET  
- Enterprise services,  
  ASP.NET, ADO.NET  
- Web services support | J2EE  
- EJB  
- JSP  
- JDBC |
| 2002 | | |
Runtime Environments Today

Compilation

Source Code → Language Compiler → Intermediate Language

Native Code Execution

Native Code

JIT Compiler

Interpreter

Interpretation
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

### Summarizing the Technologies (1)

<table>
<thead>
<tr>
<th>Application Server</th>
<th>.NET</th>
<th>Java</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>.NET Framework IIS, MTS (COM+)</td>
<td>IBM WebSphere, BEA WebLogic, others</td>
</tr>
<tr>
<td><strong>Runtime Environment</strong></td>
<td><strong>Common Language Runtime (CLR)</strong></td>
<td><strong>Java Virtual Machine (VM)</strong></td>
</tr>
<tr>
<td><strong>Standard Libraries</strong></td>
<td><strong>.NET Framework class library</strong></td>
<td><strong>J2SE, J2EE</strong></td>
</tr>
<tr>
<td><strong>GUIs</strong></td>
<td><strong>Windows Forms</strong></td>
<td><strong>Swing</strong></td>
</tr>
<tr>
<td><strong>Transactions</strong></td>
<td><strong>Enterprise Services</strong></td>
<td><strong>EJB</strong></td>
</tr>
</tbody>
</table>
Summarizing the Technologies (2)

<table>
<thead>
<tr>
<th></th>
<th>.NET</th>
<th>Java</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Web Scripting</strong></td>
<td>ASP.NET</td>
<td>JSPs</td>
</tr>
<tr>
<td><strong>Data Access</strong></td>
<td>ADO.NET</td>
<td>JDBC</td>
</tr>
<tr>
<td><strong>Small Device</strong></td>
<td>.NET Compact Framework</td>
<td>J2ME</td>
</tr>
<tr>
<td><strong>Development Tools</strong></td>
<td>Visual Studio.NET</td>
<td>IBM WebSphere Studio,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Borland JBuilder, others</td>
</tr>
<tr>
<td><strong>Web Services</strong></td>
<td>ASP.NET, .NET My Services,</td>
<td>Some support from IBM, et</td>
</tr>
<tr>
<td><strong>Support</strong></td>
<td>others</td>
<td>al., Liberty Alliance</td>
</tr>
</tbody>
</table>
Areas for Comparison: Non-Technical Issues

• Vendor issues
  – Lock-in
  – Trust
• Maturity
• Existing developer skills
  – Developer productivity
• Operating system support
• Cost
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

```csharp
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

4 classes

2 aspects

- 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
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*: 
Email: 
Address*: 

☐ I accept the license agreement which can be found here

Go to download

* Indicates required fields
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


