NEDERLANDSE JAVA USER GROUP

J-Spring
12 mei 2005

Java Is Around You

Duncan Mills
Technology Evangelist
Oracle Corp

ORACLE

The Java Vision

Java Is All Around?

- Mobile phones
- PDAs
- Wearable computers
- Real-time systems
- Mars rovers
- Embedded systems
  - No not in my fridge
  - But yes in my PVR
- But…

Corporate Development

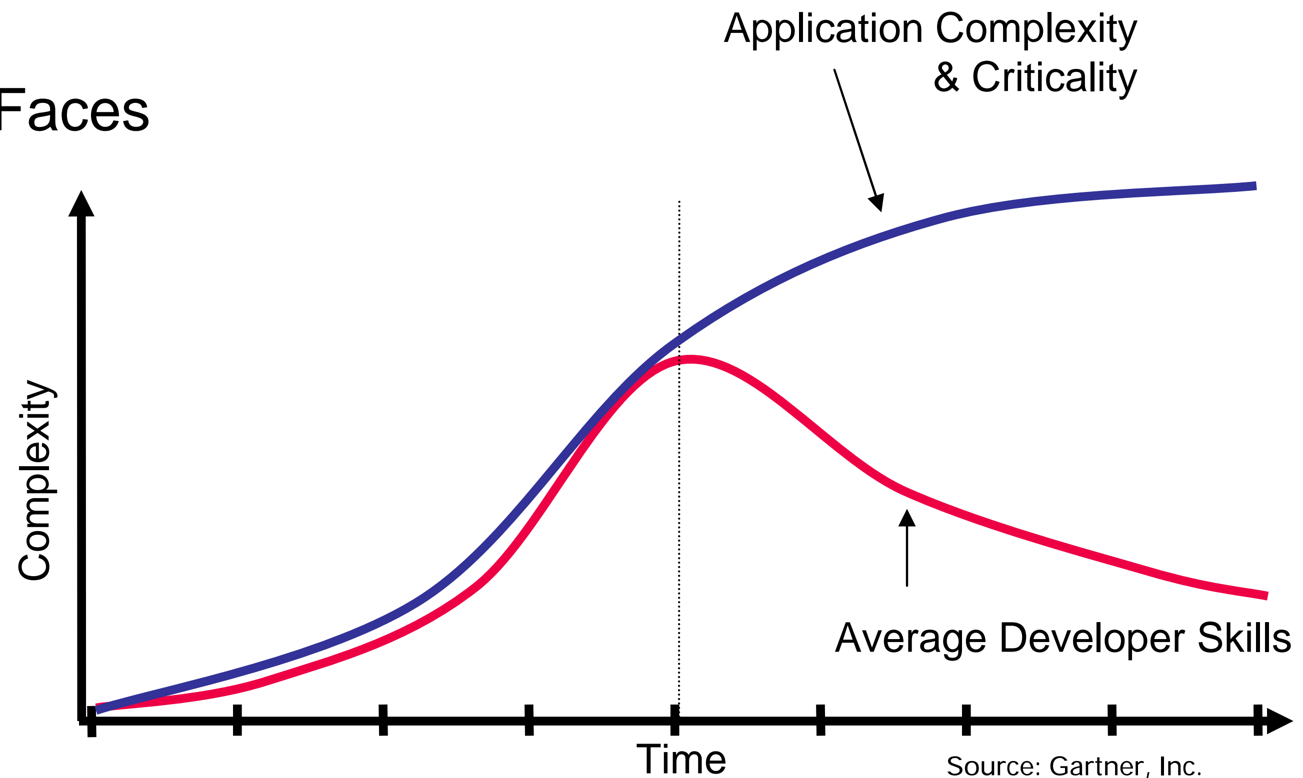
- A harder problem…
- What can we say makes Java an attractive platform?
  - Powerful yet simple
  - Rich APIs
  - Portability
  - Penetration
  - Google!

Corporate Development

- Issues
  - Lack of clarity and direction
  - Multiple standards
  - Overweight implementations
  - Rate of change
  - The Java “Community”
  - Open source
  - Productivity drop for 4gl programmers
  - Web User Interfaces

Is There Hope For Corporate Java?

- Positive Signs
  - "Year of the POJO"
  - Meta-Frameworks
  - EJB 3.0
  - JavaServer Faces



Back to Basics with POJO

- Plain Old Java Objects
  - POJOs rule persistence
    - TopLink
    - Hibernate
    - And now EJB 3.0…
  - Web Clients
    - Compare and contrast: Struts v's Tapestry or JSF
  - Benefits
    - Simplicity
    - Testability
    - Philosophy
- Enabling technologies
  - IoC / Dependency Injection
  - AOP

Meta-Frameworks

Defining a Meta-Framework

- Broad Scope
  - Choice of technologies
- Coexistence
  - Pluggability and Loose Coupling
- Abstraction
  - Supports pluggability and longevity
- Longevity
  - Critical mass / vendor support
  - Adaptable
- Tooling

Key Services / Components

- O/R mapping
  - EJB 3.0
- UI
  - JSF
- Glue
  - Providing the abstraction e.g JSR 227
- Management
  - Deployment
  - Runtime monitoring

Looking Forward…

- What's Missing
  - Security
  - Validation framework
  - Web-Flow standards
  - Runtime versioning
- Candidates
  - Spring
  - Oracle ADF
  - Apache Beehive (BEA)
  - (Keel)

EJB 3.0

EJB 3.0 Goals

- Simplify developers life
  - EJB now resembles Plain Java Object (POJO)
  - Use metadata annotations
    - XML descriptors are no longer necessary
    - Defaults
  - Unnecessary artifacts are optional
  - Simplify client view using dependency injection
- Standardize persistence API for Java platform
  - Based on success of leading ORM solutions

POJO Entity Beans

- Concrete classes (no longer abstract)
- No required interfaces
  - No required business interfaces
  - No required callback interfaces
- Support new()
- getter/setter methods
  - can contain logic (e.g., for validation, etc.)
- No exposure of instance variables outside bean class
- Use Collection interfaces for relationships
- Usable outside the EJB container
  - (Means we needed to sacrifice CMRs)
- The bean is the DTO

EJB 2.1 Entity Bean


```
public abstract class CustomerBean
     implements EntityBean {

   public abstract String getName();
   public abstract void setName (String name);
   public abstract Account getAccount();
   ...
   public String ejbCreate(String name)
     throws CreateException {
     setName(name);
     return null;
   }
   ...
}
 // Home and Component interfaces definitions,
etc.
```

EBJ 3.0 Entity Bean

```
@Entity public class Customer {
   private Long id;
   private String name;
   private Address address;
   private HashSet orders = new HashSet();

   @Id(generate=AUTO) public Long getId() {
     return id;
   }

   protected void setId (Long id) {
     this.id = id;
   }
...
```

EBJ 3.0 Entity Bean cont…

```
...

@OneToMany(cascade=ALL, mappedBy="customer")
public Set<Order> getOrders() {
  return orders;
}

public void setOrders(Set<Order> orders) {
  this.orders = orders;
}

// other business methods, etc.
}
```

Session Bean Simplifications

- Eliminated requirement for Home Interface
  - Not needed for session beans
- Business interface is a POJI
  - Bean can have more than one business interface
  - Can support remote access
  - EJB(Local)Object removed from client view
  - RemoteExceptions are removed from programmer and client view
- Eliminated requirement for unnecessary callback methods
  - Removed requirement to implement javax.ejb.SessionBean

EJB 2.1 Session Bean


```
import javax.ejb.*;
import java.rmi.RemoteException;
public interface Cart extends EJBObject
{
  public void add(String item) throws  RemoteException;
  public Collection getItems() throws  RemoteException;
  public void completeOrder() throws
                 NotInCartException, RemoteException;
}
```

```
import java.rmi.RemoteException;
import javax.ejb.*;
public interface CartHome extends EJBHome
{
  public Cart create() throws CreateException,
                              RemoteException;
}
```

EJB 2.1 Session Bean Class


```
public class CartEJB implements SessionBean{
  protected Collection items = new ArrayList();
  public void add(String item)
  {
      items.add(item);
  }
  public Collection getItems(){
      return items;
  }
  public void completeOrder(){ .. }
  public void ejbCreate(){}
  public void ejbActivate(){}
  public void ejbPassivate(){}
  public void ejbRemove(){}
  public void setSessionContext(SessionContext context){}
}
```

EJB 2.1 Deployment Descriptor


```
<session>
   <display-name>Shopping Cart</display-name>
   <ejb-name>MyCart</ejb-name>
   <home>CartHome</home>
   <remote>Cart</remote>
   <ejb-class>CartEJB</ejb-class>
   <session-type>Stateful</session-type>
   <transaction-type>Container</transaction-type>
</session>
```

Same Example: EJB 3.0

```
@Remote
public interface Cart {
    public void addItem(String item);
    public void completeOrder();
    public Collection getItems();
}
```

Same Example: EJB 3.0

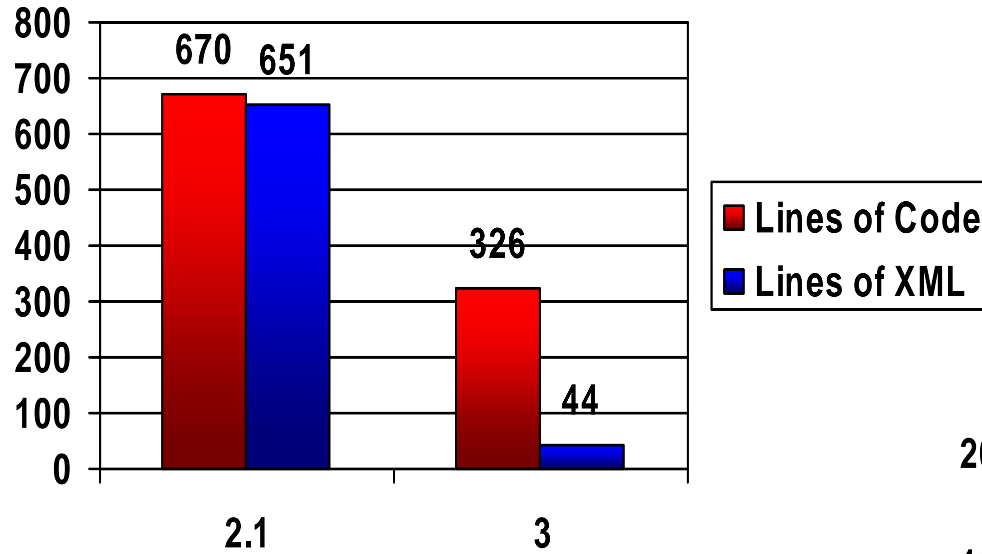
```
@Stateful
public class CartBean implements Cart {
    private ArrayList items;

    public void add(String item) {
        items.add(item);
    }

    public Collection getItems() {
        return items;
    }
   @Remove
   public void completeOrder()
   {
   }
}
```

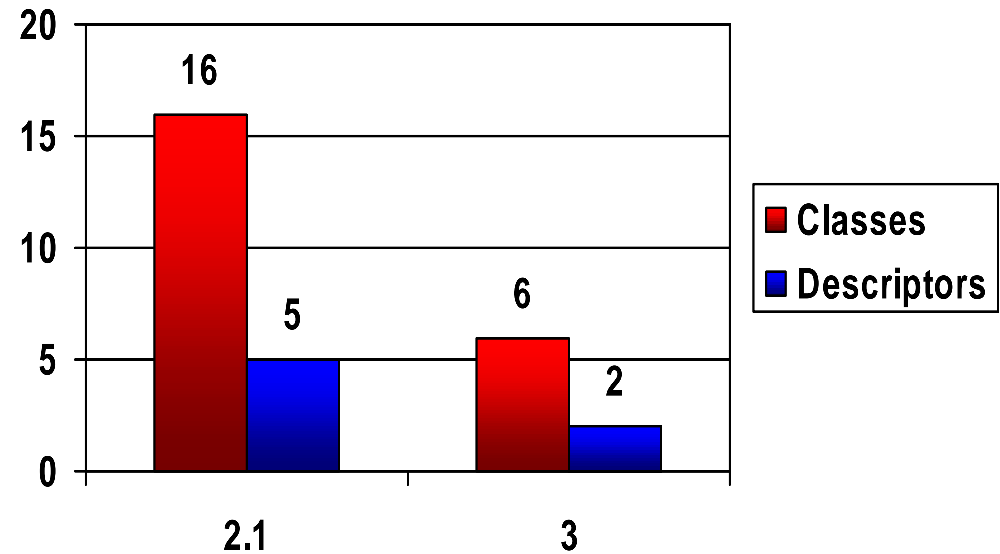
Deployment Descriptor

Simplified Access to Bean's Environment

- JNDI APIs out of developer's view
  - Not at all a good “hello world” experience
- Techniques / mechanisms
  - Declarative expression of dependencies in metadata
  - Container injection of resource entries, etc.
  - Simple programmatic lookup mechanisms
- Different usages, both have their place
  - Very simple; facilitates testability (especially setter injection techniques)
  - More flexible; dynamic

2.1 v's 3.0 Complexity





EJB 3.0 - Implications

- View of the container as a service injection facility
  - Injection points specified through metadata
  - View of the container “disappears”
- View of “component” changes:
  - Classes that require / request services
  - Pick and choose what they need
- Contractual view becomes inverted
  - Bean does not implement predefined set of APIs
  - Instead, bean specifies what it needs
- More extensible architecture => our view of “container” evolves

JSF

JavaServer Faces

- Component based UI Framework
    - Defined by JSRs 127, 252
    - Standard to be rolled into J2EE 1.5
- Multi-platform, multi-device
  - "One UI to rule them all"
- View and Controller functionality
- Rich event model
  - UI events & page lifecycle
- Extensible
  - Decorators / replacement implementations

JSF Abstraction

- Abstracts away from HTTP
  - No need to understand the request/response cycle.
- Abstracts away from HTML
  - Developers work with components not markup
  - Developers don't need to decode requests
- Implicitly future proof
  - If you are disciplined

JSP vs. JSF

	JSP	JSF
Developer	Control	Ease of Development
Events	HTTP	Java
UI	Markup	Components
Product	Pages	Applications

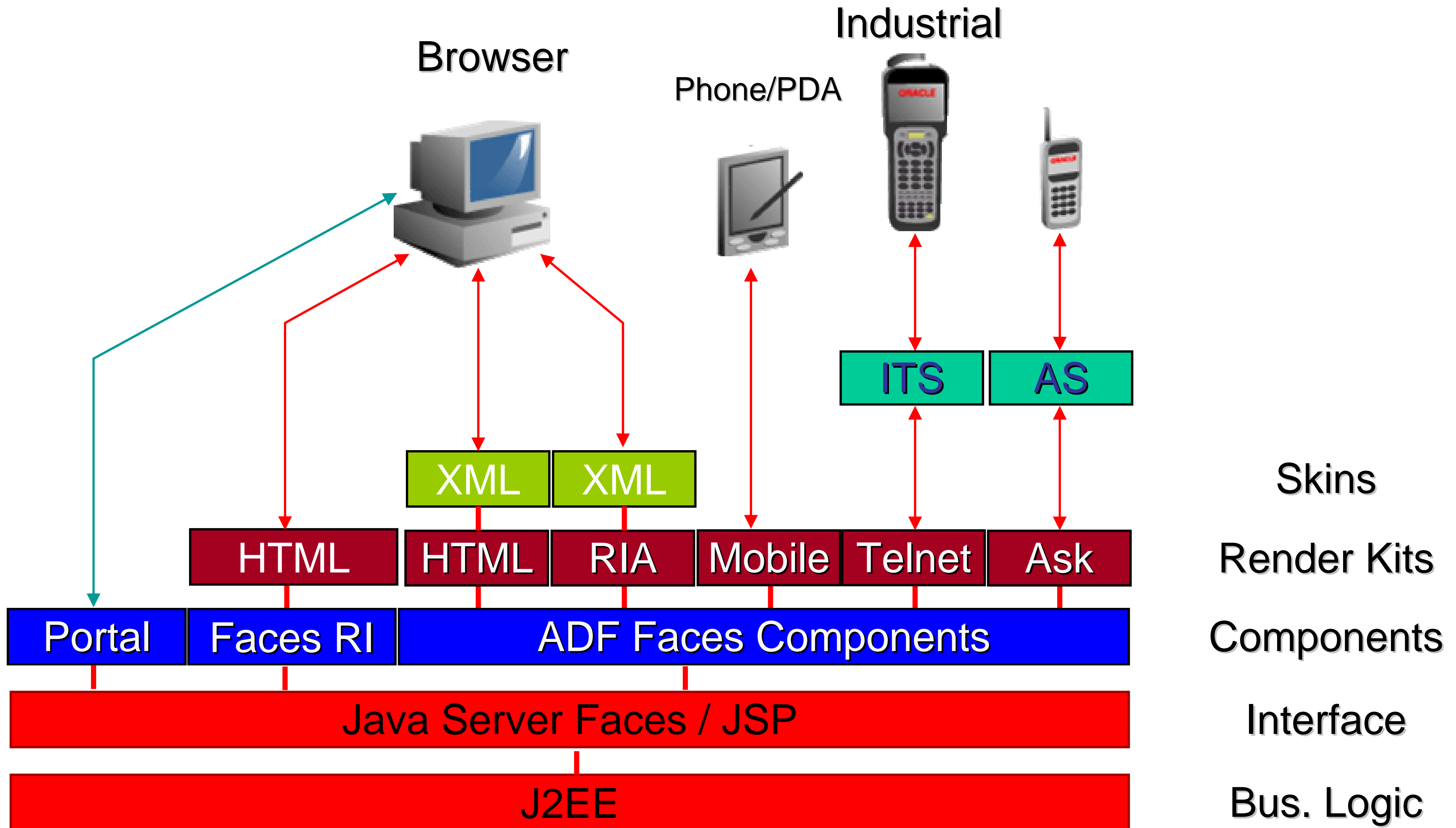
JSF Technology

- A basic JSF application consists of:
  - JSF Components – usually encapsulated in taglibs
  - Navigation Model – defines rules for navigation
  - Managed Beans - UI logic & domain objects
  - Helper Objects – Validation and conversion
- Physical terms
  - XML definition(s)
  - POJO domain objects
  - UI Beans – may be component specific
    - Listeners & Actions

JSF Component Model

- Everything is a component
  - Complex aggregate controls
    - Trees, data-grids, shuttles
  - Layouts
    - Like Swing
- JSF Components consists of three things
  - Components
    - Functionality, definition, or behavior
  - Renderers
    - Converts components to and from a specific markup language
  - Render Kits
    - Library of Renderers
    - Basic HTML RenderKit is part of the specification

JSF – Deployment Possibilities



JSF Demo

Java Is Around You

Duncan Mills
www.groundside.com/blog