I Already Use Spring, JSF, and/or Hibernate; Why Would I Use Oracle ADF?

TECH13

UKOUG Technology Conference

Presented by: John Jay King

Download this paper from: http://www.kingtraining.com

Session Objectives

- Learn how Oracle's ADF and JDeveloper may be used to create applications
- Compare Oracle ADF with available Java EE Frameworks
- Choose appropriate framework for specific tasks

Today's Agenda

- Need for Frameworks in Java EE Development
- Review of Oracle ADF, Spring, Struts, Hibernate, and JSF
- Understanding Oracle ADF tools
- Comparing and Contrasting Tools

Who Am I?

- John King – Partner. King Training Resources
- Oracle Ace Director
- Member Oak Table Network
- Providing training to Oracle and IT community for over 25 years – http://www.kingtraining.com
- "Techie" who knows Oracle, ADF, SQL, Java, and PL/SQL pretty well (along with many other topics)
- Leader in Service Oriented Architecture (SOA)
- Member of ODTUG (Oracle Development Tools User Group) Board of Directors (until Jan 1!)

Who Are You?

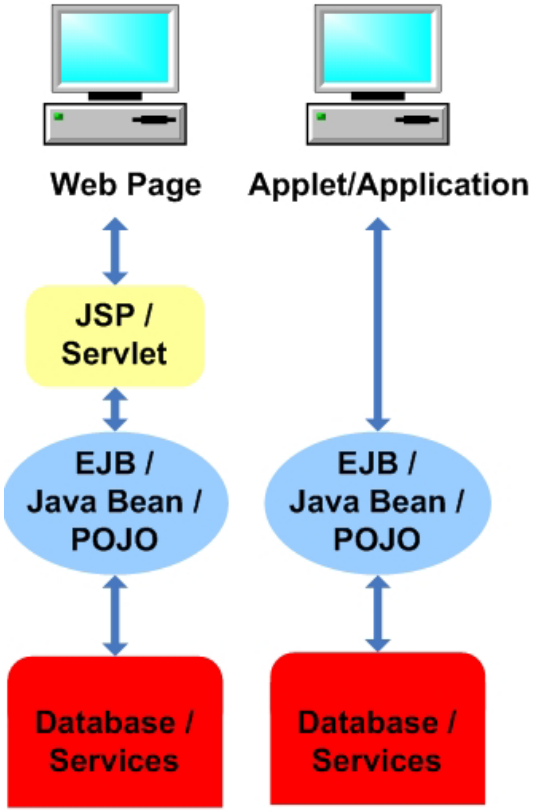
- Web Application Developer?
  - Oracle ADF?
  - Oracle APEX?
  - Oracle Forms?
  - Java?
  - .NET?
  - Other?
- Database type wondering what all of this hubbub is all about

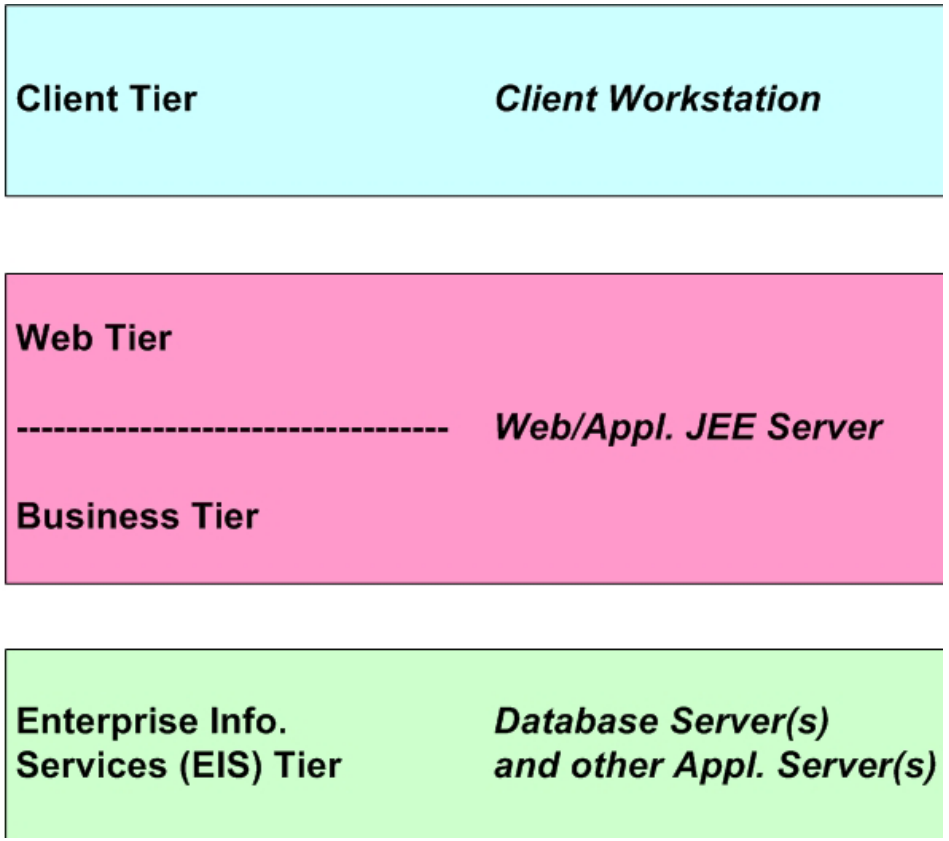
Java Web Applications

- Java-based Web Applications provide a rich and useful mechanism for allowing customers to:
  - Retrieve information (maps, searches, account retrieval, check status, etc…)
  - Create information (profiles, orders, accounts, etc…)
  - Modify information (profiles, orders, accounts, etc…)

Web Application Partitioning

- Web applications work using a variety of tools; some on the client-side and some on the server-side





Web Application Tiers

- Client Side
  - Browsers and Mobile Devices
  - Web Service Consumers
- Web/Application Java EE Server
  - HTTP
  - Jave EE
- Enterprise Tier
  - Databases
  - Web Service providers

Client-Side Tools

- Web application client-side tools provide the user interface; an important component in the overall user experience; they include:
  - HTML & HTML5
  - JavaScript
  - AJAX
  - Java Applets
  - Flash / Silverlight
  - Android / iOS
  - More…

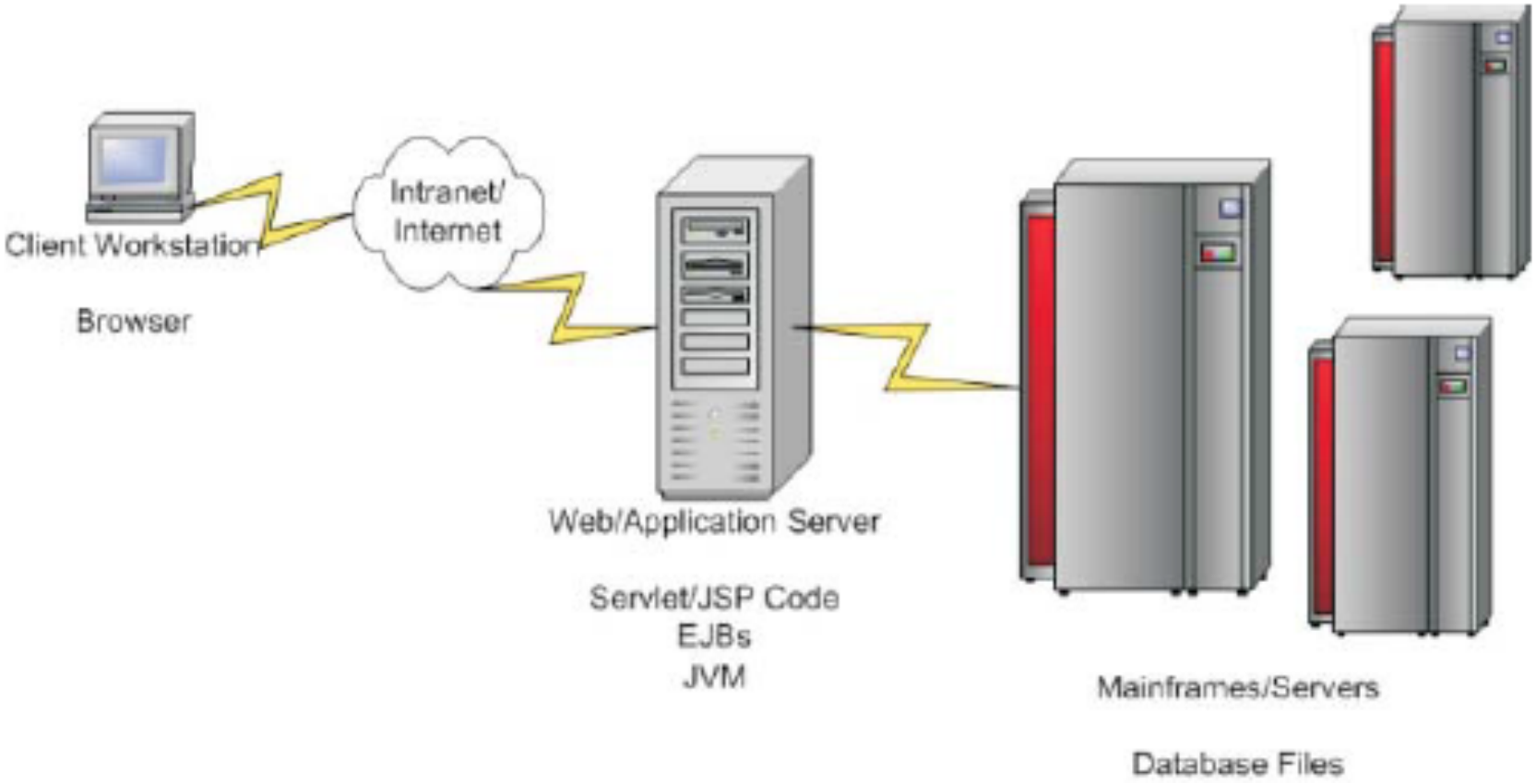
Server-Side Tools

- Java web server-side tools include:
  - Java Servlets, POJOs, JavaBeans
  - JSP, JSTL
  - Source-side JavaScript (Node.js)
  - Web Services
  - Databases and Files
  - More…

Tough to Build and Maintain

- Lots of tools, means lots of complexity
  - How to get data from client to server?
  - How to get data from server to client?
  - How to fetch/modify database data?
  - Where should work be done; client or server?
  - Where should business logic reside?
  - Which tools to use? How to use them?

What Does it Look Like?

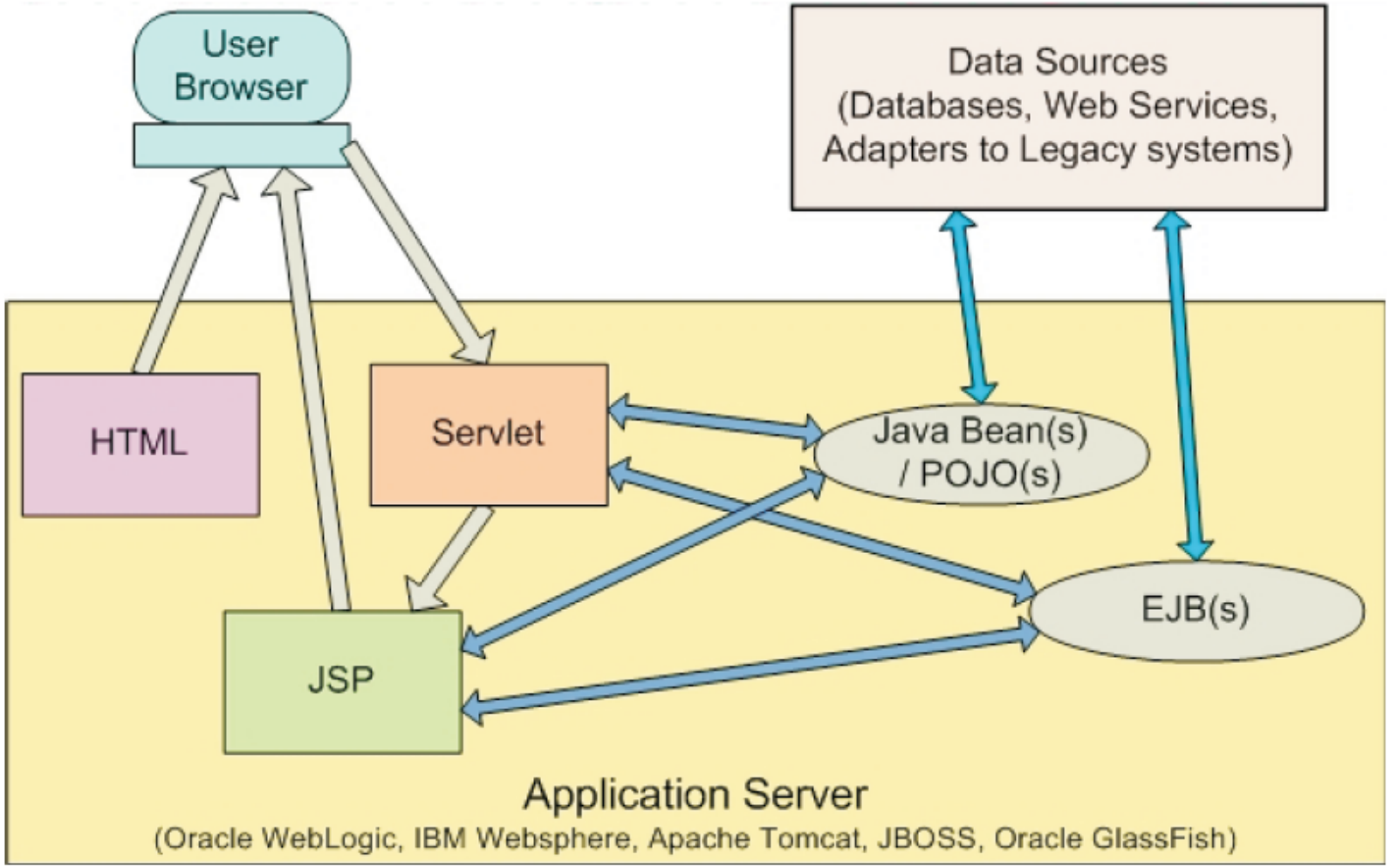


How Does it Work?

- Client-Side/Server-Side are coordinated:
  - User interface via web pages (HTML and JavaScript) allowing maximum portability
  - Servlets & JSPs are typically deployed in a Model-View-Controller (MVC) pattern
    - Servlets generally provide Controller function receiving and editing user input, & how to proceed
    - POJO/Java Beans/EJBs typically provide the Model function containing the code to access data sources based upon messages from the Controller or View
    - JSPs generally fulfill the View function providing the user interface returned to the user's browser

Model-View-Controller

- "Typical" Java EE – MVC application



Why Use MVC?

- Clearly separates logic (Controller), stored data (Model), and presentation (View)
- MVC Roles
  - Model Represents data, usually persistent (e.g. Java code interacting with data store)
  - View Represents presentation of the data (e.g. JSP/Servlet presents HTML page)
  - Controller Represents business logic and control of flow (typically a Servlet, maybe JSP)

Need for Frameworks

- The complexity of Java EE applications, especially when MVC architecture is involved requires:
  - Standardization of design and coding
  - Cooperation and sharing of code
  - Lots of coding (much of it repetitive)
  - Coordination of tasks
  - Opportunity for “cowboy developers” to put code where it does not belong

Home-Grown Frameworks

- In the early days of Java EE (then called J2EE) many shops “rolled-their-own” and created frameworks to serve the needs of their shop providing standardization, coordination, and libraries of reusable code
- As time progressed these home-grown frameworks became a major source of maintenance and enhancement efforts
- Most shops don’t want to be in the software tool business!

"Open Source" Frameworks

- Open Source is great
- Open Source is free ("like a puppy, not like beer")
- Many Open Source frameworks exist today
  - Some are completely Open Source
  - Some are Open Source technically but are managed strongly by vendors
  - Open Source projects often turn sharply and different versions might be incompatible

Framework Proliferation

- Frameworks are a useful tool; most shops use more than one:
  - Client-side frameworks (jQuery, Angular.js, Ember.js, etc…)
  - Data access (usually ORM) frameworks (Hibernate, Spring, JPA, TopLink, etc…)
  - MVC frameworks (Struts, Struts 2, Spring MVC, JSF, Tapestry, Wicket, etc…)

ORM Frameworks

- Java is an Object-Oriented language and its datatypes do not directly translate into relation data types
- Object-Relational Mapping frameworks allow Java programmers to read and modify database data using standard Java rather than SQL via JDBC
- The most common ORM frameworks are Hibernate, Spring, and JPA

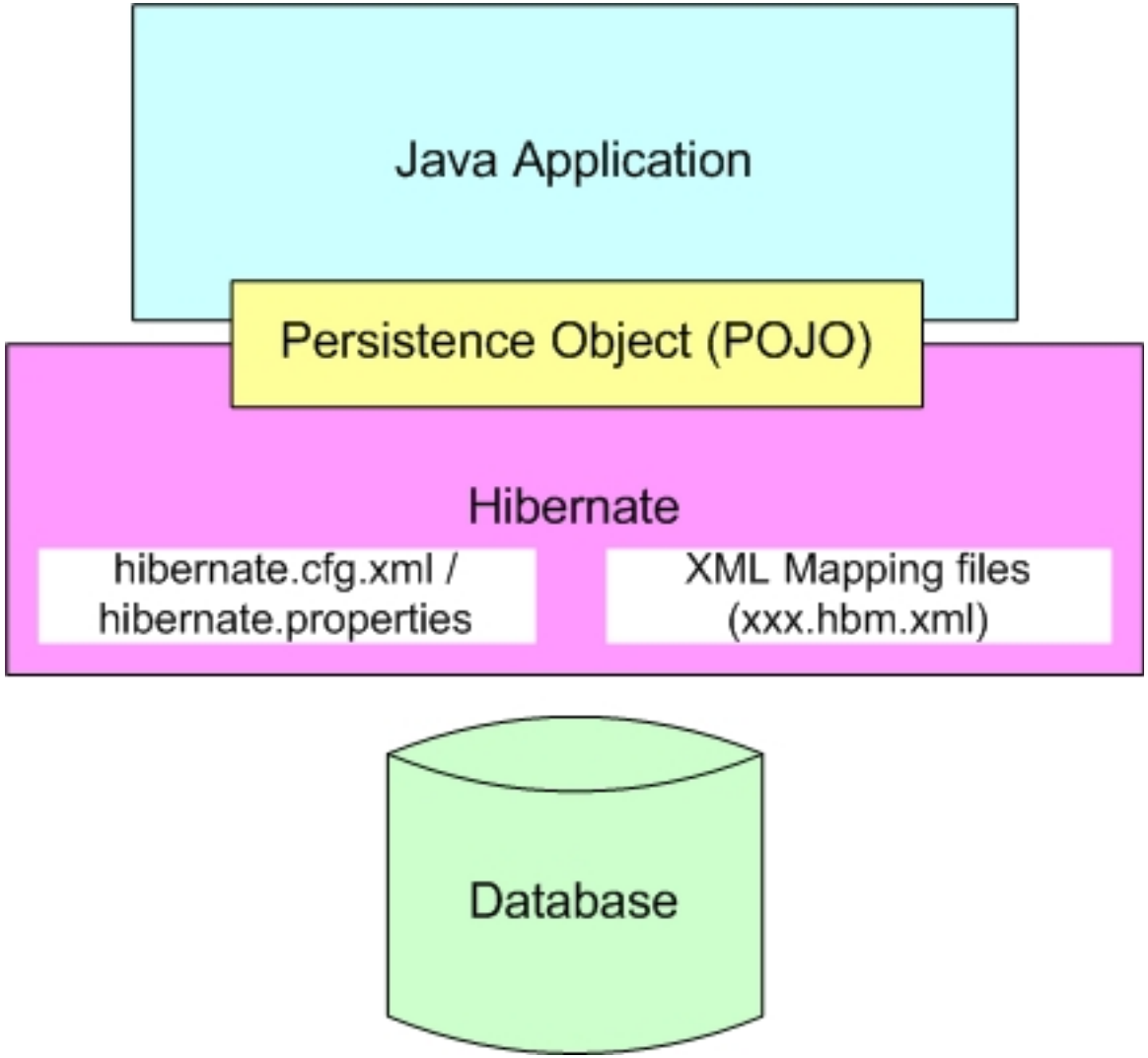
Hibernate and JPA

- Hibernate is an Open Source framework managed by Red Hat
- Hibernate was so successful that much of EJB 3 and JPA are patterned after it; in turn, Hibernate morphed to become compatible with JPA

How Does Hibernate Work?

- Hibernate is configured at two levels:
  - Application level defining database & connection
  - Java object level; mapping Java objects (POJOs) to database table(s)
- Early Hibernate required use of configuration via property files; later releases used XML configuration files; and the latest versions allow the use of Java annotations rather than configuration files

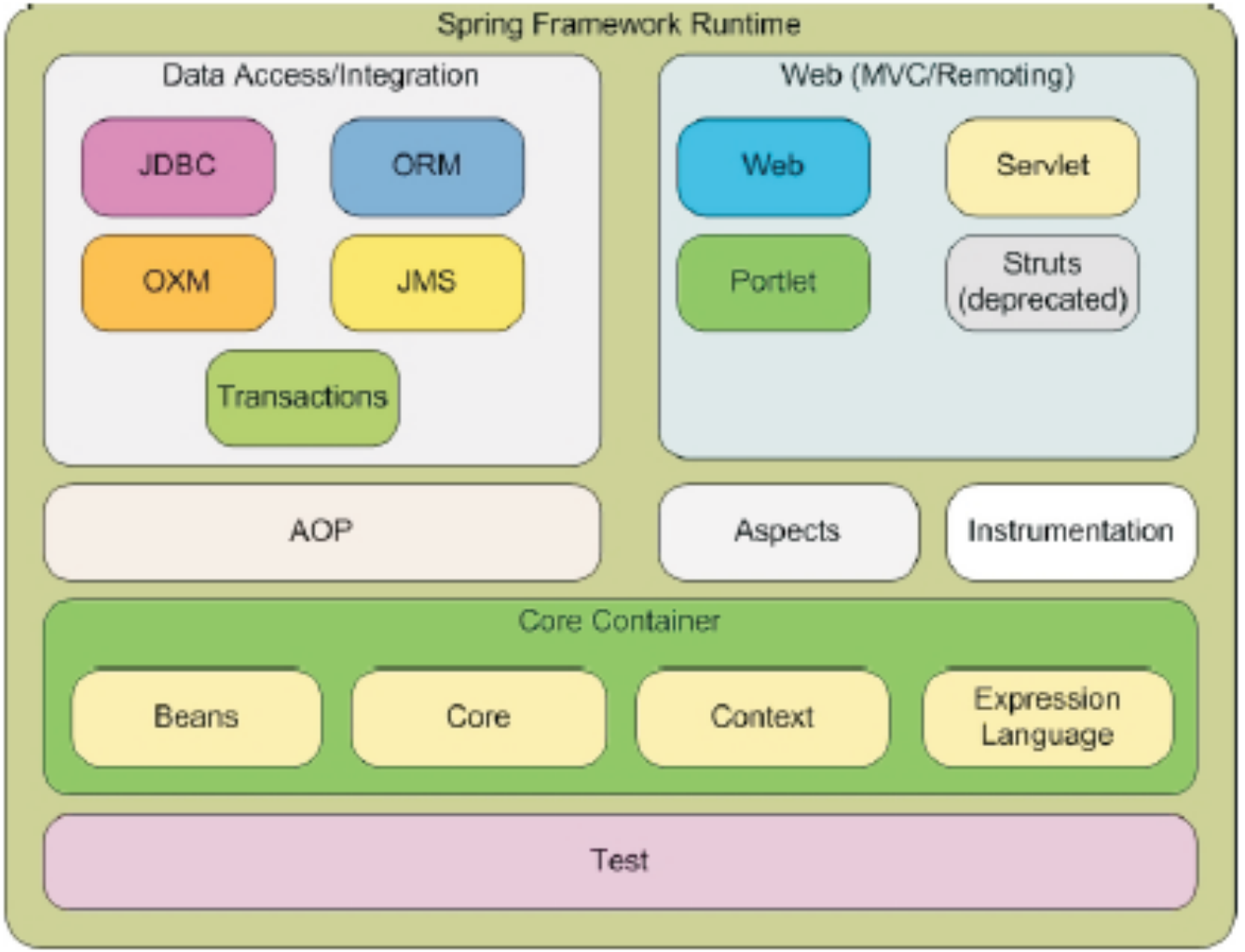
Hibernate Components



Spring

- Spring Framework provides many mechanisms to simplify the Java including Spring MVC (discussed later)
- Spring's Dependency Injection allows the use of Java objects without dependency
- Spring's wide-open design makes data access possible using Spring's own persistence and DAO mechanisms; or, ORM using Hibernate, JPA, or just about anything else

Spring Architecture



Spring Configuration

- Configuration in early versions of Spring was all done with XML
- Today, it is also possible to provide much of Spring's configuration using Java annotations

Java MVC Frameworks

- Perhaps the greatest number of Java frameworks is available for MVC including:
  - Struts and Struts 2
  - Spring MVC
  - Wicket
  - Tapestry
  - Vaadin
  - JSF
  - MyFaces
  - Stripes
  - More (including non-Java options)

Struts and Struts 2

- Apache Struts (aka Jakarta Struts) is the grand-daddy of Java MVC frameworks
  - Struts provides its own Controller Servlet
  - Form-bean, Action, and other objects are used to work with the View
  - Struts may use Model Java Beans/EJBs
- The Struts framework is designed to help developers create MVC web apps
  - "request" handler maps to a URL
  - "response" handlers transfers control
  - Tag library used by developers

Spring MVC

- Spring MVC is open but structured and based upon DI and IoC
- Like Struts, Spring provides the controller
- Spring MVC integrates well with existing frameworks like Struts and ORM tools like Hibernate
- Less-complex than some other frameworks; easy to test
- XML configuration files and Java annotations are used for configuration

Java Server Faces (JSF)

- JavaServer Faces (JSF) was conceived by the Java Community Process (JCP) group to provide a standardized GUI (Graphical User Interface) that can be used by any framework
- JSF developers build web applications by
  - Assembling reusable UI components in a page
  - Connecting these components to an application data source
  - Tying client-generated events to server-side event handlers

How Does JSF Work?

- JSF includes:
  - APIs for UI components, their state, handling events input validation, and page navigation (also internationalization and accessibility)
  - A JSP custom tag library implementing the JavaServer Faces interface
  - Many custom classes some representing HTML objects and others representing UI actions and events
  - XML configuration files
  - JSF "life cycle" for executing applications

So, Which Is Better?

- All of them
- None of them
- It depends (correct answer…)
- Real questions are:
  - What are you trying to do?
  - Which tools fit best?

Industry Ratings

- Industry ratings indicate that:
  - Struts is easy to learn but can be limiting
  - Spring MVC is a little more-difficult to learn but provides greater flexibility
  - JSF is the most difficult of the major Java MVC tools to learn but also has the greatest flexibility
- Generally, the greater the flexibility the bigger (and slower) the footprint

IDEs

- A variety of Integrated Development Environments (IDEs) are in use including:
  - Non-IDE editors like vi, emacs, UltraEdit, etc…
  - Eclipse (including IBM RAD) has largest market share; used for non-Java development via plug-ins too
  - IntelliJ (proprietary) smaller market share but used by “the cool kids”
  - JDeveloper (proprietary) smaller market share; core tool for all Oracle tools
  - NetBeans (proprietary) used mostly by students; reference tool for latest Java versions

IDEs and Frameworks

- All of the major IDEs provide support for development in Struts, Spring, and JSF; however, much of it is manual
- SpringSource provides an Eclipse-based tool for creation of Spring MVC applications
- IBM RAD (Eclipse) provides wizards for creating JSF applications and advanced support for Struts
- Only marginal IDE support is available for JavaScript libraries and Ajax (thought DOJO is partly supported by RAD)

Still Too Complex

- My real point is:
  - Whichever set of tools you select, you must find a way to make them work together
  - Does your presentation layer provide the rich client interface your customers demand?
  - Does your model layer work well with your database? Do your people know how to use it?
  - Does your controller layer provide flexibility?
  - Is work occurring in the wrong places?
- How much time and energy do you want to expend coordinating disparate software?

Oracle ADF

- Oracle's Application Development Framework is not a traditional Java framework; I call it a "meta-framework"
- ADF combines frameworks into one tool:
  - ADF BC
    Model layer works with many databases, stored procedures, and web services
  - ADF Faces
    Presentation layer including rich-client interface
  - ADF Faces Controller
    Controller improving upon JSF capabilities

Oracle JDeveloper

- Oracle's JDeveloper IDE has a patchy history but today is quite good:
  - First version purchased from Borland and heavily modified; never successful
  - Oracle rewrote JDeveloper using Java; better design but implementation was not good
  - JDeveloper 11 and now version 12 provide a world-class IDE with complete support for MVC
    - Business Components
    - ADF Faces (JSF) Components including Rich-Client interface (Oracle JavaScript libraries & Ajax)
    - ADF Controller task flows

Oracle Eclipse

- Oracle Enterprise Pack for Eclipse (OEPE) provides the ability to create and maintain Oracle ADF applications using Eclipse (and Eclipse-based tools) rather than JDeveloper
- JDeveloper is Oracle's primary tool so new features are supported first (and probably best) in Jdeveloper before being ported to Eclipse
- Oracle's NetBeans tool is not set up for ADF

JDeveloper: One More Thing

- By focusing development efforts on JDeveloper; we now have a simple platform for:
  - ADF and Java development
  - Business Rules
  - SOA
  - BPEL
  - more …

How Does ADF Work?

- Thousands of Java classes make up the ADF class libraries
- XML configuration files are used to declaratively provide control of ADF objects and execution
  (managed via IDE wizards and editors)
- JDeveloper allows developers to extend the ADF class libraries and provide designer code at every level
- ADF objects may be packaged into .jar files allowing easy reuse of definitions and code

ADF BC

- Business Components provide access to data via service components for:
  - Database data (Oracle and others) and stored procedure calls
  - Web Services calls
  - “Flat Files”
- ADF-BC database access is performed using JDBC and an ORM tool
- Objects are defined declaratively; XML configuration is automated but may be modified by the developer

ADF Faces - View

- ADF Faces is an Oracle extension of Java Server Faces (JSF)
- Objects are defined declaratively or "painted" via editors; again, Java may be used to customize the built-in Java classes
- ADF Faces includes built-in JavaScript and AJAX components providing client-side editing and rich-client interaction (via libraries so that when underlying technology changes, applications do not)

ADF Faces – Controller

- ADF application MVC flow is controlled using a set of “task flows”
  - Task Flows are defined using a “painter” style editor and/or XML configuration files
  - Task Flows, like other ADF objects may be packaged and reused in multiple applications

Oracle ADF Mobile

- ADF Mobile uses the standard JDeveloper environment but has special form-factor objects and support for touch interactions
- Once a mobile application is developed it may be deployed to both Android and iOS
- ADF's framework obscures specific interactions with the user device and may be updated "under the covers" without changing existing applications
  (for example; Oracle replaced the original PhoneGap code with Cordova last year)

Oracle ADF and Java EE

- Oracle ADF is fully Java EE compliant
  - The Oracle ADF class libraries are part of Oracle WebLogic installations at no additional charge
  - Oracle ADF may also be deployed to Oracle's Glassfish server in test environments
  - Oracle ADF may be deployed to non-Oracle servers (e.g. Tomcat, WebSphere, Jboss) but Oracle will charge a licensing fee for deployment

Comparing & Contrasting

- All of the major Java EE tools have strong followings; so, how do you evaluate?
  - What do you need to do?
  - Do you have existing code or developer skills?
  - Does your installation already have the WebLogic Server for some reason?
  - Does solution meet Java standards?
  - Does your IDE support the rich-client, model, and MVC frameworks?
  - Is your MVC framework really MVC?
  - How much work is required integrate tools?

Major Points: ADF BC

- If you have lots of Oracle background, ADF-BC will look and feel familiar; in addition you have abilitity to control SQL
- If you have Java-only expertise, the ADF-BC interface may seem complex; but may actually be simpler than “wiring” mechanism in other ORM tools
- Supports stored procedures and web services better than most other tools
- Once created, ADF-BC objects may be shared across applications via .jar files

Major Points: ADF Faces

- Rich client "baked in" – no worries on integration; however, Oracle's libraries are large and might not always include the latest in JavaScript/Ajax capabilities
- Skinning provides ability to create consistent look and feel; however, may not be as easy to fine-tune appearance of screens
- Oracle provides conflicting suggestions to not use your own JavaScript and CSS but also suggests it in some cases ????

Major Points: ADF Task Flows

- By extending the JSF flow capability ADF provides unprecedented power and capability in managing the MVC environment without “breaking” the MVC
- Though seemingly simple, ADF Task Flows continue to be a major area of learning for ADF developers
- ADF Task Flows may be encapsulated and reused

Oracle ADF Synopsis

- One packaged solution including:
  - Rich client interface (no other JavaScript framework needed)
  - Point-and-shoot GUI wizard creation of user interface (no other MVC framework needed)
  - Model layer via ADF-BC using standard Java under the covers (no other ORM needed)
- Everything starts declaratively using wizards; however, at any time Java may be extended as desired

Conclusion

- Not really a conclusion; I hope this session acts more as a beginning of discussion
- If you have existing investments with Java EE frameworks; the inertia required to shift to ADF for new projects may be too much
- If you do not have existing investments in Java EE frameworks; you might want to give a long look at a single-unified environment

www.rmoug.org

COLLABORATE 14 – IOUG Forum

April 6 – 10, 2014

The Venetian Las Vegas, NV

ODTUG Kscope14

SEATTLE, WASHINGTON · JUNE 22-26

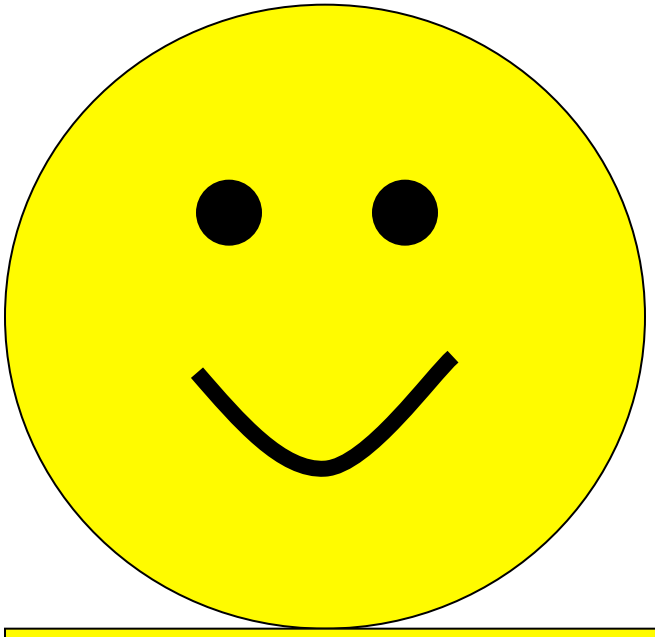
REGISTER TODAY!

TOPICS:

Application Express | ADF and Fusion Dev. | Developer's Toolkit | The Database
Building Better Software | Business Intelligence | Essbase | Planning | Financial Close
EPM Reporting | EPM Foundations and Data Management | EPM Business Content

www.kscope14.com

Please Complete Session Evaluations



Thanks for your attention!

I Already Use
Spring, JSF, and/or Hibernate;
Why Would I Use Oracle ADF?

To contact the author:

John King

King Training Resources

P. O. Box 1780

Scottsdale, AZ 85252 USA

1.800.252.0652 - 1.303.798.5727

Email: john@kingtraining.com

Today's slides and examples are on the web:

http://www.kingtraining.com