

ORACLE®

THE **INFORMATION** COMPANY

ORACLE

Duncan Mills
J2EE Evangelist
Oracle Corporation

ORACLE

A New Face for Java

An Introduction to JSF

ORACLE

Agenda

- What is JSF and where does it come from?
- Architecture
- The ecosystem

ORACLE

What is JavaServer Faces™

- A standard J2EE framework for generating thin client user Interfaces.
 - Not just HTML pages
- Defined through the Java Community Process
 - JSR-127 – 1.0/1.1 May 2001 → May 2004
 - JSR-252 – 1.2 Aug 2004 → (Aug 2005)
- Standard support as part of JEE 1.5
- In J2EE 1.4 implemented as a Servlet
 - Multiple Implementations

ORACLE

To start, a bit of background and history. JSF is currently undergoing it's 1.2 revision, with the specification for that in public draft now. 1.2 does not actually make many changes to the core 1.1 specification but concentrates on integration issues when running JSF in a JSP environment. There is no time to get into those issues here and everything we cover will apply equally to 1.2 as well as 1.1.

The core contributors to the standard were Sun, Oracle and IBM (the actual members of the expert group can be viewed on the JCP website). The birth of the standard was long and troubled but the end result was without compromise and worthwhile. As we'll see later, the standard was not developed in a vacuum, many of the key concepts and best practices were adopted from the existing J2EE framework ecosystem.

At the moment there are two significant implementations of the specification the Sun RI (reference implementation) and the Apache MyFaces project. These have both passed the TCK.

The Elevator Pitch

- Simplifying Web Development
 - Component based not mark-up
 - Automatic event and state handling
 - Diverse client base not just HTML
 - Designed with tooling in mind
 - Applicable to wide spectrum of programmer types
- Oh and it's a J2EE standard

ORACLE

Given 30 seconds to sell the concept of JSF to a developer, what are the key messages?

First and foremost JSF is about Components in the traditional UI sense. JSF is not slaved to the traditional mark-up centric view of Servlets and JSP. A button is a button, not a tag. It has behaviour, you can listen for it being pressed, all of that infrastructure is managed for you. As a programmer you don't really care that you are running on the web. In fact JSP does not enforce HTML Browsers as a client. The client could be a portal, WAP browser, a handheld, or something more exotic as we'll see later on.

Both the nature of this programming model and the through that was devoted to "tool-ability" during the design if JSF makes it a suitable foundation for both Rapid Application Development (RAD) environments, appealing to the Business Programmer* and RAD developers, as well as more fully featured code-centric IDEs. In fact this tool-ability is still a focus of the standards body through JSR-276 – Design Time Metadata for JSF Components (Ref:

<http://www.jcp.org/en/jsr/detail?id=276>) which aims to ensure that components from different vendors can be correct rendered in every IDE that supports JSF.

But JSF is not a "nobby" framework. Although the basic concepts are simple and approachable, the expert programmer has an immense amount of flexibility and power over the way that the framework operates. Everything is pluggable.

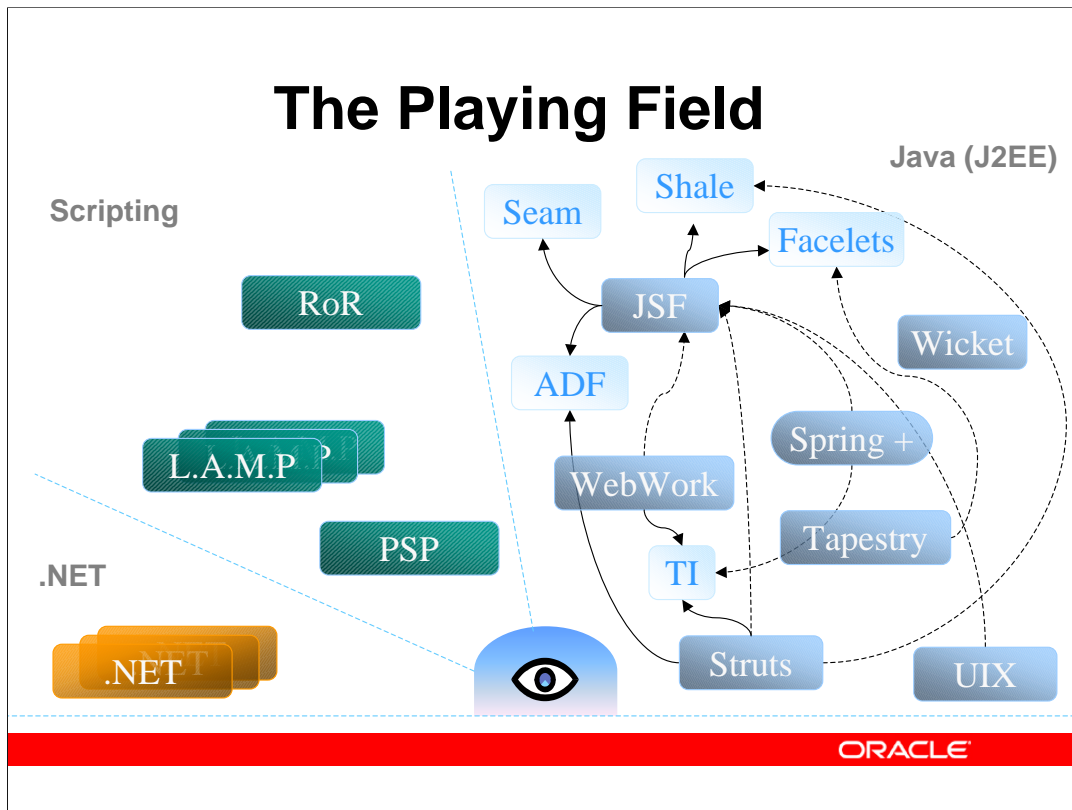
**Generally defined as a programmer being primarily focused in a business domain rather than say strict OO principles. Think the expert user in a department who builds complex Excel macros to handle that year end reporting drudgery..*

Questions and Misconceptions...

- Do we really need yet another Web Framework?
- I've heard that it's way too complex
- I've got to use a tool don't I?
- Oh is it out yet?

ORACLE

When JSF first hit the streets there was a great deal of FUD flying around with respect to what it actually meant. Above are some of the common questions and misconceptions that you will still hear today, although often confined to programmers who either feel that Emacs was the pinnacle of tool evolution, or who have a stake in alternative frameworks, of which there are many and which we'll discuss in a moment.



The choice of technology when it comes to building Web apps is huge. In this slide I've just tried to give an impression of some of the options, based around various zones. With .NET of course one has a single runtime platform with multiple implementations in terms of languages but no real choice within those language choices.

The scripting world has a vast array of options, I've highlighted a few in general here.

PSP – PL/SQL Server Pages – Pages generated by Oracle PL/SQL procedural code. A very popular implementation choice for Oracle shops.

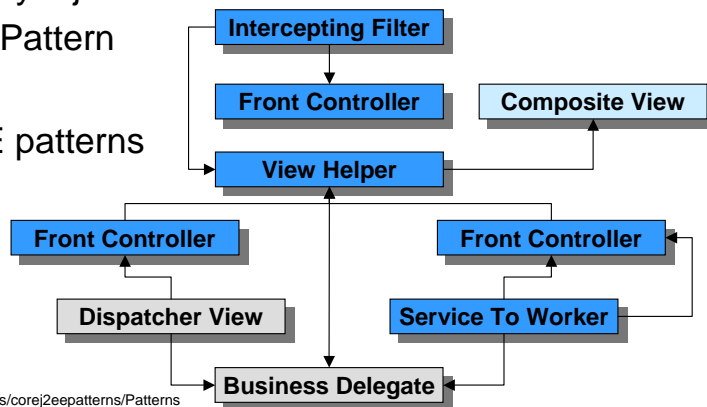
L.A.M.P – covers a multitude of sins standing for amongst other things Linux, Apache, MySQL and Perl. The acronym having a broader meaning now to any free and open source stack. As such we're not dealing with a development framework here so much as an architecture.

RoR – Ruby on Rails – The hot new kid on the block

In the J2EE space we have tremendous variety from the venerable Struts framework that held the torch for so long up to brand new frameworks like Wicket, covering every extreme from mark-up centric to class centric to XML centric and everything in between. We can trace some of the trends and ideas propagate through the Petri disk, tracing the ancestry of some of the key ideas within JSF to earlier frameworks and seeing how even JSF is evolving to spawn sub-frameworks or meld into meta-frameworks, adopting new and old ideas in the process. The J2EE space is certainly maturing rapidly and we're beginning to see consolidation – such as the recent WebWork + Struts merger into Struts TI/(Struts Actions).

Second Generation Framework

- JSF adopts several key concepts
 - Dependency Injection
 - Decorator Pattern
 - POJOs
 - Core J2EE patterns



Source: <http://java.sun.com/blueprints/corej2eepatterns/Patterns>

ORACLE

As explained on the previous slide, the ancestry of JSF can be traced back to several existing frameworks in the Java Web Space. We see a definite evolution in best practice thinking from the granddaddy of them all, Struts, through to JSF. The modern mantras adopted by JSF are.

- 1) Don't lookup resources inject them – the adoption of a lightweight version of dependency injection will certainly invoke thoughts of Spring although it is no-where as comprehensive as the latter framework. (As an interesting sidebar on this particular topic see JavaOne 2005 session TS-5068 - Spring and JavaServer Faces Technology: Synergy or Superfluous? Johnson / Mills. <http://developers.sun.com/learning/javaoneonline/2005/webtier/TS-5068.html>).
- We also see injection creeping into other modern frameworks such as the EJB 3.0 specification.
- 2) Don't Subclass – Decorate. Add functionality in a cumulative and non invasive mannner – quasi-AOP without the AOP if you will
- 3) Plain Old Java Objects. Again in contrast to Struts where the Form beans used as backing for the UI had to implement a particular interface, the managed beans that fulfil the same function for JSF are POJOs giving a much looser coupling, improving testability and so on. We see the POJO "pattern" as one of the key architecture forces over the last two years, first Spring, then JSF and now EJB. (ref: The Rise of the POJO; http://www.oracle.com/technology/tech/java/newsletter/articles/rise_of_the_pojo.html)

(Continued...)

- 4) Core J2EE patterns. The Sun Blueprints site publishes a set of core J2EE patterns and pattern macros. I have reproduced a portion of the overview diagram in the side, highlighting the patterns used by JSF. For information I have included brief descriptions of each pattern on the diagram below, the full description of the patterns can be found at the URL mentioned in the slide.
- First and foremost we have the **Front Controller** pattern. As we'll see later the whole of the JSF model is based around an events model which JSF as a Front Controller intercepts, decodes and dispatches on the basis of.
 - The **Intercepting Filter** pattern is implemented through the pluggable decorator mechanisms that JSF provides for customising the lifecycle and many other aspects of the JSF milieu such a expression language resolution or navigation handling.
 - The **Composite View** pattern is not directly implemented by JSF, although the potential is there and the Facelets project looks to plug that niche in a way much better suited to JSF than say Tiles (Ref: Facelets Project: <https://facelets.dev.java.net/>)
 - The concept of **View Helper** is key to any modern framework. The Page mark-up itself contains little or no logic and certainly no `<%script%>` tags. As we'll see JSF is based around a component model with a very rich set of tags / components that fit nicely into this pattern.
 - The **Service To Worker** pattern is one of two macro patterns which (along with Dispatcher view) describes a composition of the dispatcher (e.g. Navigation Controller) function and content creation function. JSF falls within the Service To Worker umbrella when we look at how much work the framework does based both on the handling of navigation events and the potential for view construction inherent in the framework through the viewHandler mechanism (more later)
 - **Business Delegate** is not really a pattern that JSF implements directly, so I've not highlighted this function. A Managed Bean could fulfil this role, but in doing so it would generally need some outside help generally from a meta-framework such as Spring or Oracle ADF, to actually manage those delegate beans. JSR-227 (Ref: <http://www.jcp.org/en/jsr/detail?id=227>) fills this particular gap in a generic way.

Useful Pattern Definitions

Intercepting Filter

Create pluggable filters to process common services in a standard manner without requiring changes to core request processing code. The filters intercept incoming requests and outgoing responses, allowing preprocessing and post-processing. We are able to add and remove these filters unobtrusively, without requiring changes to our existing code.

(Continued ...)

Front Controller

Use a controller as the initial point of contact for handling a request. The controller manages the handling of the request, including invoking security services such as authentication and authorization, delegating business processing, managing the choice of an appropriate view, handling errors, and managing the selection of content creation strategies.

Composite View

Use composite views that are composed of multiple atomic subviews. Each component of the template may be included dynamically into the whole and the layout of the page may be managed independently of the content.

View Helper

A view contains formatting code, delegating its processing responsibilities to its helper classes, implemented as JavaBeans or custom tags. Helpers also store the view's intermediate data model and serve as business data adapters.

Dispatcher View

Combine a controller and dispatcher with views and helpers to handle client requests and prepare a dynamic presentation as the response. Controllers do not delegate content retrieval to helpers, because these activities are deferred to the time of view processing. A dispatcher is responsible for view management and navigation and can be encapsulated either within a controller, a view, or a separate component.

Service to Worker

Combine a controller and dispatcher with views and helpers to handle client requests and prepare a dynamic presentation as the response. Controllers delegate content retrieval to helpers, which manage the population of the intermediate model for the view. A dispatcher is responsible for view management and navigation and can be encapsulated either within a controller or a separate component.

Business Delegate

Use a Business Delegate to reduce coupling between presentation-tier clients and business services. The Business Delegate hides the underlying implementation details of the business service, such as lookup and access details of the EJB architecture.

Architecture

ORACLE

Ok at the high level we know that JSF is all about building User Interfaces for the web – what makes it tick? Why is it different?

Key Terms

- **UI Component**
 - JSF is component based
- **Managed Bean**
 - Objects maintained by the JSF inversion of control mechanism
- **Expression Language**
 - The ties that bind
- **Navigation Case**
 - The rules that govern page flow
- **Lifecycle**
 - The guts of the thing

ORACLE

Before we get into the nitty-gritty of Lifecycle which we'll cover last, let's concentrate on the key objects and concepts that that you have to understand to get into JSF. We'll go through these one by one over the next few slides....

Components Are King

- A page definition is (potentially) abstract
 - Constructed from components and containers (also components)
 - Conceptually similar to the Swing / AWT model
- Components have:
 - Attributes
 - Behaviours
 - One or more renderers (Components themselves are implementation independent)
- Components may be nested

ORACLE

As we saw before, the concept of a page being constructed using components, rather than a series of mark-up tags, is key to JSF. The programmer is dealing with an entity that has properties which can be set in the page definition, or indeed in code. Likewise the component will have behaviours and the coding paradigm is based on the JavaBeans model here. If a component has a value-changed event I just create a listener for that event. The framework will then invoke that listener at the right time, As the programmer I don't have to get involved with decoding the request to work out if code needs to be executed or not*.

Components are standardised within JSF, you can use any component from any vendor with the framework and mix and match them as required, they all behave in the same way.

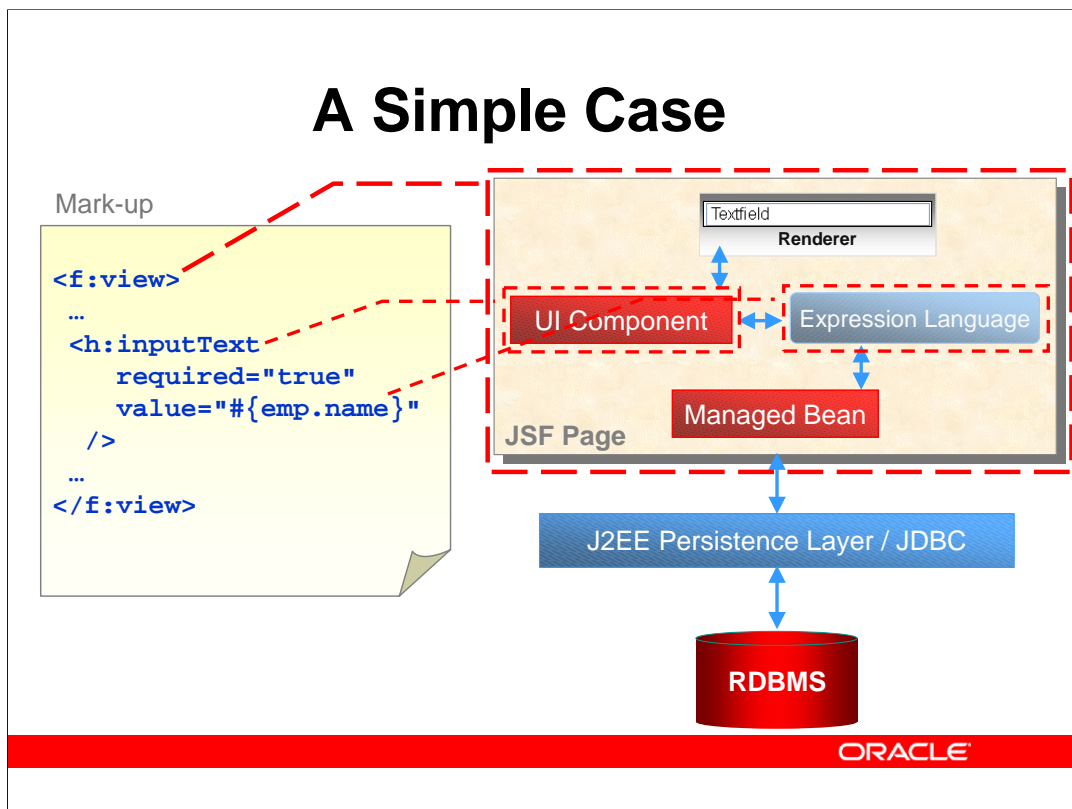
The component itself has key attributes such as the read-only attribute or width attribute, but it says nothing about how the component will look. That is is responsibility of the renderer which is part of the component definition, but not something that the programmer has to deal with. It is the job of the renderer, at render time to realise the component as defined as a concrete screen instruction – e.g. in the case of a browser application – a fragment of HTML mark-up. A renderer for a component may output very complex html (take a tree control component as an example of this), but the programmer is insulated from that. The generated mark-up may include JavaScript as well as HTML if a particular component has complex client side behaviours. Renderers also insulate the programmer from the minor differences between browsers, particularly important where scripting is involved.

(Continued ...)

Component nesting allows complex screen layouts to be achieved (some of these containers are analogous to Swing layout managers) , but also allows for situations where child components inherit behaviour from their parent. An example of the latter is a table component which will contain multiple children. Tables support various range related (and in some cases Sorting) events which need to propagate to and influence the values of nested children.

The final twist in the component story is in fact one of the most powerful features, although rather advanced. The design-time page definition is translated into a component tree at run time. This tree is an object structure that can be manipulated like any other, so the programmer has the potential to dynamically add, remove and change tree nodes to totally customise the page contents. This is vital for handling customisation and personalisation features.

** However, as a programmer I may be interested in understanding the JSF lifecycle so I can see the context in which an event is executing.*



As a concrete example let's look at very simple page (I've omitted a little detail here for clarity), We have a `<f: view>` tag and then an `<h:inputText>` within it. That seems simple enough but there is a little more going on.

Firstly this looks a little like JSP, I though it was JSF? Well that's because it is JSP. Today the implementations of JSF happen to use JSP as a carrier technology. That is, the page definitions are JSP pages and the JSF components are expressed as JSP tag libraries. This however is a convenience, it means that conventional HTML / JSP editors and IDEs can be used to design JSF pages. JSF itself makes no assumptions about the format of the page definition, that is just one of those pluggable facets of JSF. A pure XML page description, or even a binary format would all be acceptable, it doesn't matter.

So in our little example we have an input item of some sort, it seems to have a property "required" set to true, this is where the behaviour of the component is being defined, although there is no mention of how that will be enforced. Depending on the renderer, it may be implemented client side using JavaScript if the component is smart enough.

Then there is this value attribute it is set to an expression `#{emp.name}`. This is a reference to some kind of model object (in Model-View-Controller terms) . We'll talk about expressions shortly.

If we expand that simple piece of mark-up into a resolved view we see some of the important players in JSF.

1. The view tag encapsulates the whole page definition and contains the component tree.

(Continued ...)

2. The `inputText` tag corresponds to the `UIComponent` in the tree
3. The `#{}` expression maps into a managed bean (more later) which in most systems would in turn access some persisted data perhaps ultimately in a relational database

The renderer is not defined by the page at all, how the component actually gets output will depend on the user agent (e.g. the Browser type) and perhaps the look and feel if the components in question support "Skinning", as for instance the ADF Faces components do (Ref: <http://otn.oracle.com/jsf>).

D E M O N S T R A T I O N

A Basic JSF Page

ORACLE

In this demonstration we'll look at the process of basic page creation and event handling. Illustrating the following:

1. Basic page creation
2. Adding components to the page
3. Action handling
4. Setting Component properties at runtime.

Steps (Assuming JDeveloper – although Sun Studio Creator will do just as well):

1. Create empty page. Enable autobinding – we'll assume the page managed bean (backing bean) is called page from now on
2. Drop a `inputText`, `commandButton` and `outputText` into the page. Set the label property of the Button – this could also be an expression – it does not have to be literal
3. Those components will appear sequentially on the same line – what if I want to have a different layout? Drop in a `panelGrid` – set the column Number to 1
4. Move the components into the Grid – layout will now be vertical. (Of course much more powerful layout components are available)
5. Now to implement a little code to copy the contents of the input field into the output field with a suitable message

(Continued ...)

```
public String commandButton1_action() {  
    String name =  
    (String)getInputText1().getValue();  
    getOutputText1().setValue("Hello " + name);  
    return null;  
}
```

This code simply gets the current value of one component and then copies it with a little decoration into the other. – We'll cover the "return null" in a moment.

What was significant about that?

- I did not have to decode the HTTP request to get the value the user entered
- I did not have to work out that the user had pressed a particular button – my event code was invoked by the framework
- I did not have to play with the response to write in the new output value – I simply set the property on the relative component.

Managed Beans

- Java Objects (empty constructor), Maps, Lists
- Defined in faces-config.xml
- Defined with various scopes
 - application
 - session
 - request
 - none
- Lazy initialisation by JSF as needed

ORACLE

Along with components, managed beans are a key part of JSF. When we created the page in the demo just now we automatically created a managed bean (AKA a Backing Bean) which held references to the objects we created on the page and provided a place to put code.

Note: JSF does not mandate that every page has to have its components defined as Java objects in a managed bean. A page can happily live without any backing bean at all, or a backing bean can only contain references to a few of the components on the page. In most cases the backing bean will only contain action code and no references to the UI Components – unless you need to programmatically manipulate them.

The lifecycle of a managed bean is handled by JSF – when the bean is first referenced, JSF will create an instance of the relevant Java class and keep it around as long as is specified by the bean definition. The **none** scope above is used to define transitory objects that are just used to populate other managed beans and is never used outside of that context.

Any Java object can be a managed bean as long as it has a no-arg constructor. It is also possible to rip out this default IoC mechanism and plug in an alternative bean management engine. The common case for this is to plug in the Spring container as it has a much richer model for managing beans through factories etc as well as direct instantiation.

Managed Bean Example

- Definition

```
<managed-bean>
  <managed-bean-name>userbean</managed-bean-name>
  <managed-bean-class>com.oracle.sample.User</managed-bean-class>
  <managed-bean-scope>request</managed-bean-scope>
</managed-bean>
```

- Usage in a page

```
<h:inputText value="#{userbean.name}"/>
```

- Usage in code

```
Application app = FacesContext.getCurrentInstance().getApplication();
ValueBinding bind = app.createValueBinding("#{userbean}");
User user = (User)bind.getValue(ctx);
```

ORACLE

So the managed bean is defined in XML within the faces-config.xml file. This file exists in the WEB-INF directory of the application and is the master file for a JSF app.

As we can see the basic definition of a bean is pretty simple, we provide an alias "userbean" in this case, the actual class to instantiate "com.oracle.sample.User" and the lifespan; request in this case.

Usage of that managed bean is then through Expression Language which is key to JSF and something we cover next.

Expression Language

- Evolution



- Dot notation for attributes (JavaBean model)

- `#{userbean.name} == instance.getName();`

- Map notation

- `#{foo['baa']} == instance.get('baa');`

- Expressions can be any depth

- `#{foo['baa'].person.name}`

- Expressions can evaluate

- `#{userbean.name == foo['baa'].person.name} → true|false`

ORACLE

Expression Language (or EL) is the glue in JSF. It provides the hook between the user interface and the managed bean facility. Pretty much any property in a component can be set to an EL expression.

Obviously the primary use of this is to bind the value of a component to something in the model (as we saw earlier), but we can also bind attributes (such as the Rendered attribute supported by all components) to EL, either to set a value directly or perhaps to evaluate the expression to do a little conditional processing.

As well as binding to managed beans EL can also be used to directly reference various know scopes such as `requestScope`, `sessionScope` etc. What is more you can customize the EL resolver in JSF and plug in your own scopes. A nice example of this is the JSF-Security project (ref: <http://jsf-security.sourceforge.net>) which adds a whole new scope called *securityScope* to the expression language giving the user access to security attributes such as role from within EL. This is possible because of the extensive use of the Decorator pattern in JSF which allows developers to add functionality around the core without having to create specialised sub-classed versions of the framework.

D E M O N S T R A T I O N

Using EL

ORACLE

Building on top of the last example we'll add a little EL to the page to do some conditional operations:

1. If the output field is set disable the input field and button to prevent the user from setting the field value again

Steps (Assuming JDeveloper – although Sun Studio Creator will do just as well):

1. Open the last page
2. Set the value property of the outputText == null
3. Set the Read-only property of inputText1 to `#{ !empty page.outputText1.value }`
4. Set the Rendered property of the button to `#{ !page.inputText1.readonly }`
5. Run

What was significant about that?

- First time around we wrote code to manipulate the UI – it turns out that we can actually do an awful lot in EL
- The button depends on the inputField which in turn depends on an attribute of the outputText which is after it in the page. This is significant The component tree is completely build in memory, all of the properties and so on resolved, then and only then is the tree rendered. This means that components on the page can have interdependencies regardless of their position.

EL Elsewhere...

- Managed Properties

```
<managed-bean>
  <managed-bean-name>userbean</managed-bean-name>
  <managed-bean-class>com.oracle.sample.User</managed-bean-class>
  <managed-bean-scope>request</managed-bean-scope>
  <managed-property>
    <property-name>userType</property-name>
    <value>#{reference['USER_TYPES']['EMPLOYEE']}</value>
  </managed-property>
</managed-bean>
```

- Usage in code

```
Application app = FacesContext.getCurrentInstance().getApplication();
ValueBinding bind = app.createValueBinding("#{userbean}");
User user = (User)bind.getValue(ctx);
```

ORACLE

Although the primary use for EL is in the page definitions themselves, you can also use EL in seeding managed bean properties, or in Java code as we saw before. In the case of managed properties the dependencies will cascade during the resolution process. So in this example, as the userbean is first referenced, JSF will set the managed property for userType (calling the .setUserType() method on the User object instance), but in order to get the value to set it will also either resolve, or instantiate a new instance of the reference bean (definition not shown here).

There are scoping rules which define what scopes of managed bean are visible to other managed beans – for instance a bean defined in the session scope is visible to a request scope bean, but not visible to an application scope bean.

As a final note on Expression language we have to look at the bad side as well as the good.

Yes EL provides us with a nice un-typed abstraction for object references. It is convenient to use and very powerful. However, this abstraction is in itself a weakness. EL is meta-data and it's not compiled at design time. Consequently a mistype in an expression is enough to break the binding and cause a runtime error.

This is perhaps this issue that worries Java programmers the most, but in defence, when using an IDE such as JDeveloper, the compilation process should encompass and validate the meta-data source files as well as the Java source and verify that EL references are (as far as possible) valid.

The Navigation Model

- Command Items can raise specialised "Action Events"
 - These have an **outcome**
- Navigation in JSF is based on outcomes
- Navigation modified by
 - The view raising the event
 - The method raising the event

ORACLE

Action events are raised by components such as buttons and links – they are distinct from "normal" events such as value-change, in that they complete with a navigation outcome. A Null or unrecognised outcome means stay on the same page – we saw this in the first demo. In the faces-config you can define navigation meta-data which maps outcomes to navigation through navigation cases. Cases can be global to a page, for instance, any link that raises a "help" outcome navigates to help.jsp. Alternatively the rules can be a little more strict and the name of the action raising the outcome taken into account. Thus, two links could both raise the "help" outcome, but cause navigation to different places should the programmer desire it. In practice this second level of granularity is little used.

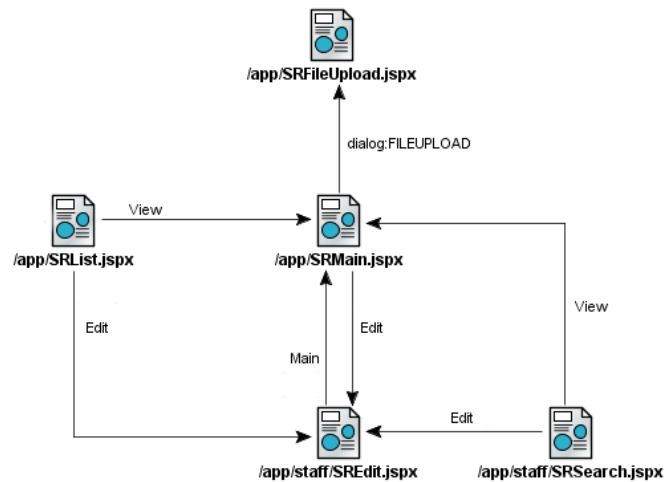
A typical navigation rule looks like this:

```
<navigation-rule>
  <from-view-id>/app/SRCreate.jsp</from-view-id>
  <navigation-case>
    <from-outcome>Continue</from-outcome>
    <to-view-id>/app/SRCreateConfirm.jsp</to-view-id>
  </navigation-case>
```

...

There is no restriction as to the number of rules that can be defined as navigation cases for a particular page.

An Example...



ORACLE

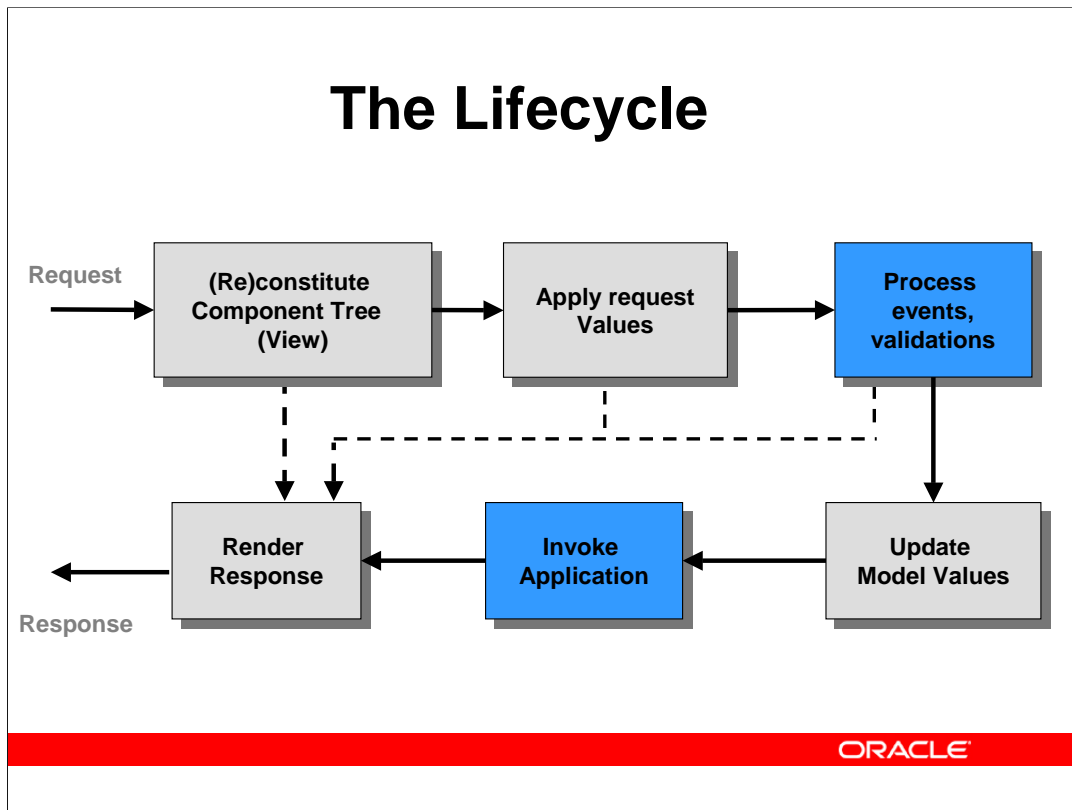
Ok so this is a visualisation of a navigation model – the real thing is just XML as we saw in the previous slide's notes, and not very interesting. However, from this diagram we can see there are a series of rules / named events linking pages (or views in JSF parlance) together. If a command item such as a button on the SRMain page in the centre here, raises an "Edit" outcome then the JSF framework will, acting as a page flow controller, forward (or redirect if required) the browser the SREdit page.

This is an example of where good tooling for JSF can make a real difference, by rendering the navigation model as a diagram (and allowing editing through that), the developer has both a productive way of defining the application page flow and as a handy side effect, a visual document of the application page structure and relationships.

If the application flow needs to change, for instance a confirmation page needs to be added, then the relationships in this diagram can be re-drawn, but the pages themselves do not have to be touched – This is an implementation of so called (JSP) Model 2 Architecture as pioneered by the Struts framework.

What is not shown on the diagram is "wildcard navigation". JSF does not force you to define every possible event. If you have common navigation events – for instance "Logout", you can use wildcards in the <from-view-id> definition basically saying, "Any page that matches this pattern can implement this outcome". The wildcard is *.

Another interesting aspect of this diagram is the use of a "dialog:" prefix on one of the rules. Basic JSF does not have provision for pop-up windows as an intrinsic part of the navigation model, however add on component sets and extensions such as ADF Faces and Shale add this capability using this kind of naming convention.



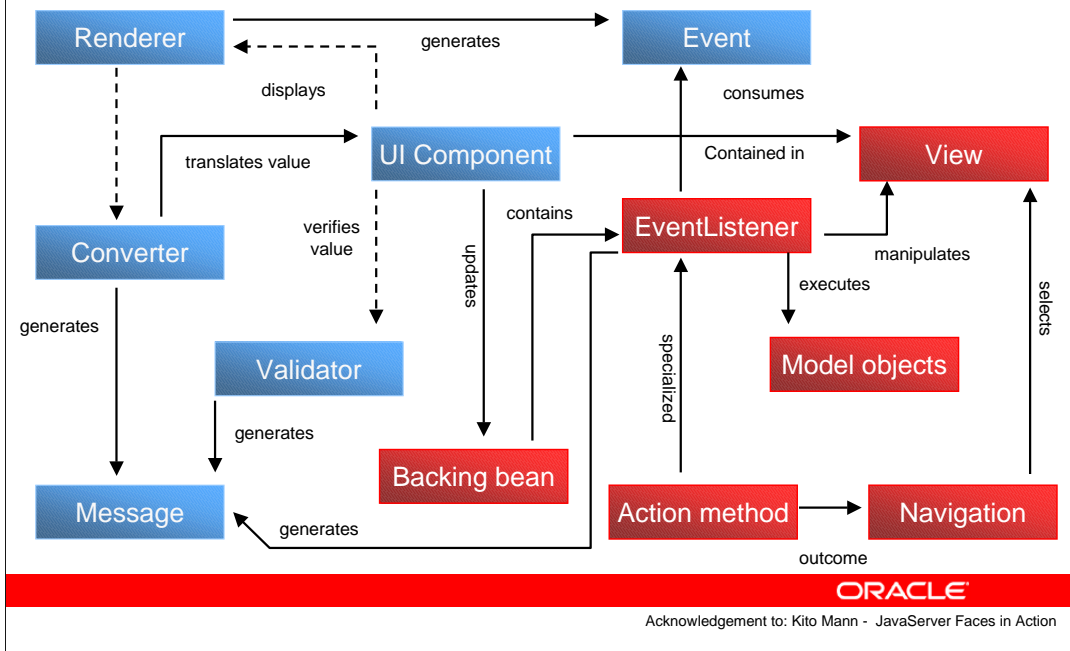
Understanding the lifecycle is important later on, but for now it is more a matter of understanding that one exists and It goes through various phases. The programmer can attach listeners to these lifecycle phases as well (remember the intercepting filter pattern?)

The key phases from our day to day perspective are Process Events and Invoke Application.

- **Process Events** is where JSF validators kick in. These are declarative validations that can be associated with components analogous to the commons validation framework. If validation fails control will return to the screen before the model (the backend data model) is updated, usually with a suitable error message. Some events marked as "immediate" also trigger this path bypassing the rest of the lifecycle processing.

- **Invoke Application** where a programmers code is actually called. By this point in time, the components have been populated and validated and the back end model updated, so the programmer knows the context in which the code is executing.

Summary of Key Elements



This diagram is a little scary but it is useful in that it shows all of the key components in the JSF architecture that we've discussed. In everyday operation, the programmer will only be concerned with the red (darker) boxes. These are the primary hook points for code. Everything on the Blue side is generally handled by the framework although as in all aspects of JSF it can be extended or customised.

We've seen how code can be added to Backing Beans (managed Beans) to define components and implement actions and listeners and how those actions are used to control the navigation of the application page flow. The view here is the page definition itself and the model represents the backend integration usually handled by managed beans.

The World of JSF

ORACLE

What about the ecosystems around JSF – who's doing this stuff?

Ecosystem

- IDEs
 - Oracle – JDeveloper
 - Sun – Studio Creator
 - Exadel – Eclipse
 - WebTools – Eclipse
 - IBM - Rational Application Developer
- Implementations
 - Sun RI
 - Apache MyFaces project
 - Glassfish (Sun) JSF 1.2

ORACLE

D E M O N S T R A T I O N

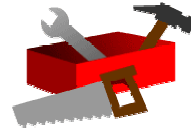
IDEs In Action

ORACLE

We've already had a taste of basic JSF programming, in this demo we'll take a more realistic view and actually bind to some real data and do some transactional operations. The point of this is to give you an idea of the potential that a good IDE can offer to boost the productivity of a JSF developer.

Components

- Several component libraries available
 - MyFaces Cherokee
 - ADF Faces
 - Tobago
 - ...
- AJAX is the next big thing
 - Smart components + asynchronous communication with server
 - JSF is ideal for this



ORACLE

The key to the success of JSF as an ecosystem will largely lie with it's ability to be a single skill that programmers can use in every situation. To realise this, the components need to be there as well. A vast range of components can only help towards this goal, in the same way that VBX and ActiveX fostered the adoption of Visual Basic by concealing complexity and making complex UI construction relatively simple.

Today JSFCentral lists 27 component vendors / projects out there. Some are extremely large sets of components like ADF Faces which contains over 100, some are small but highly specialised such as some of the charting and Graphics components.

The growth area right now is AJAX (Asynchronous JavaScript And XML). This is basically the ability for pages to communicate in an asynchronous manner with the server, for instance, updating lists on the fly based on what you are typing into a field. AJAX as a technique aims to break the application out from the constraints of the browser request / response cycle when nothing happens until the user Submits the page. The aim being to provide a more "desktop" style of interaction. Google has been leading the way in showing what can be done with this technology with Gmail and Google Suggest.

But... AJAX is not some simple library to use, it's more a lexicon of techniques and scripting, certainly not a tool for your everyday business programmer. This is where JSF has a real advantage. Any AJAX functionality (and the scripting that underlies it) is the responsibility of the component and it's renderers. The consumer of the component can just pick it up an use it, regardless of the underlying complexity.

D E M O N S T R A T I O N

Multi-Channel Deployment

ORACLE

As a finale we'll look to the really cool potential of JSF and it's components:

- Rich internet applications using AJAX enabled components to give a desktop like feel to browser deployed applications
- A blast from the past – JSF running on a telnet device – not as wacky as it might seem, many hostile environments such as warehouse floors need applications that will run on VT based terminal devices. With the right render-kits, JSF can be used to build those applications as well.
- A vision of the future. Another example of an unusual render kit is the ASK client – Generating User interfaces for asynchronous devices such as SMS and instant messenger protocols

Summary

- A modern component based framework
 - Extensible in many ways
- A mix of code and meta-data
- Ideally suited, but not confined to RAD
- Safe skills to invest in; applicable to:
 - Multiple client types
 - Technology shifts
- Still evolving; Work still needs to be done
 - Navigation Model / Security integration

ORACLE

We've covered a lot over the last 30 slides or so which I've attempted to summarise here. The key messages to take away are the flexibility of the framework and the fact that a single set of skills acquired to create a browser based application today can be applied to much more exotic media tomorrow.

JSF however, is not static, there are still issues to be addressed in the 1.3 or 2.0 timeframe. Specifically navigation and nested processes (with all of the state management implications of that) and security implementation which is certainly an area where things should be simpler out of the box.

Further Reading

- Many good books
 - JSF in Action - Manning
 - Core JavaServer Faces – Prentice Hall
- Sites:
 - jsfcentral.com → Community site
 - otn.oracle.com/jsf → Oracle JSF resource site
 - <http://java.sun.com/j2ee/javaserverfaces/> → Sun

ORACLE

Q & A

ORACLE

ORACLE®

