
ORACLE®

ORACLE
FUSION MIDDLEWARE

ORACLE

Oracle ADF – Programming Best Practices

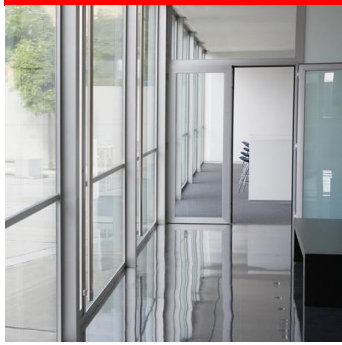
Frank Nimphius
Oracle Application Development Tools Product Management

Secret of Oracle ADF Rockstar Programmers

"You cannot buy experience, you have to earn it"

- Duncan Mills

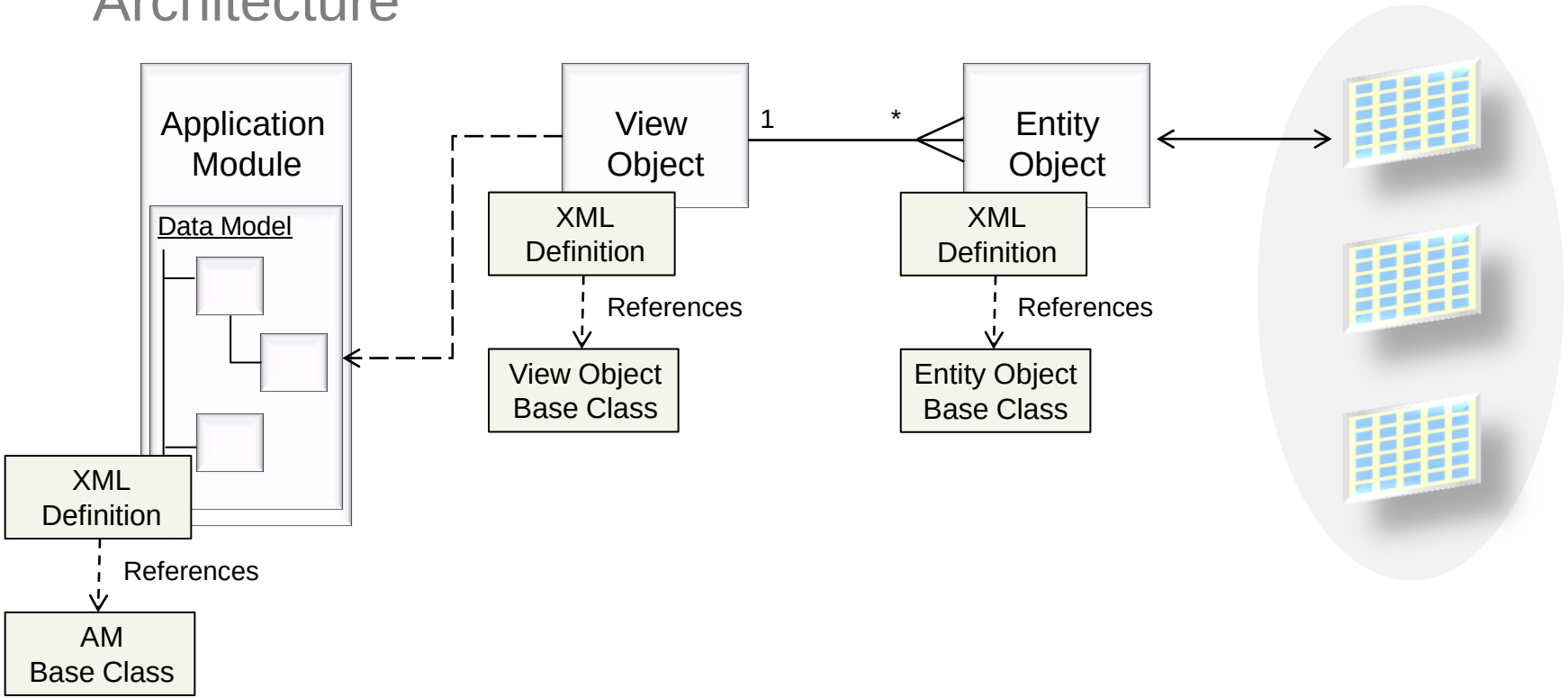
Program Agenda



- ADF Business Components
- ADF Binding Layer
- ADF Controller
- ADF Faces
- JavaScript

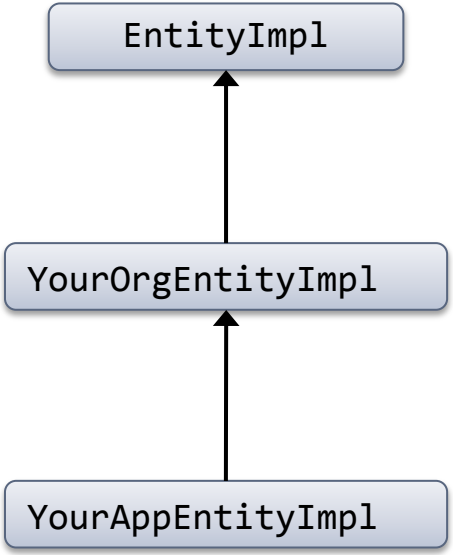
ADF Business Components

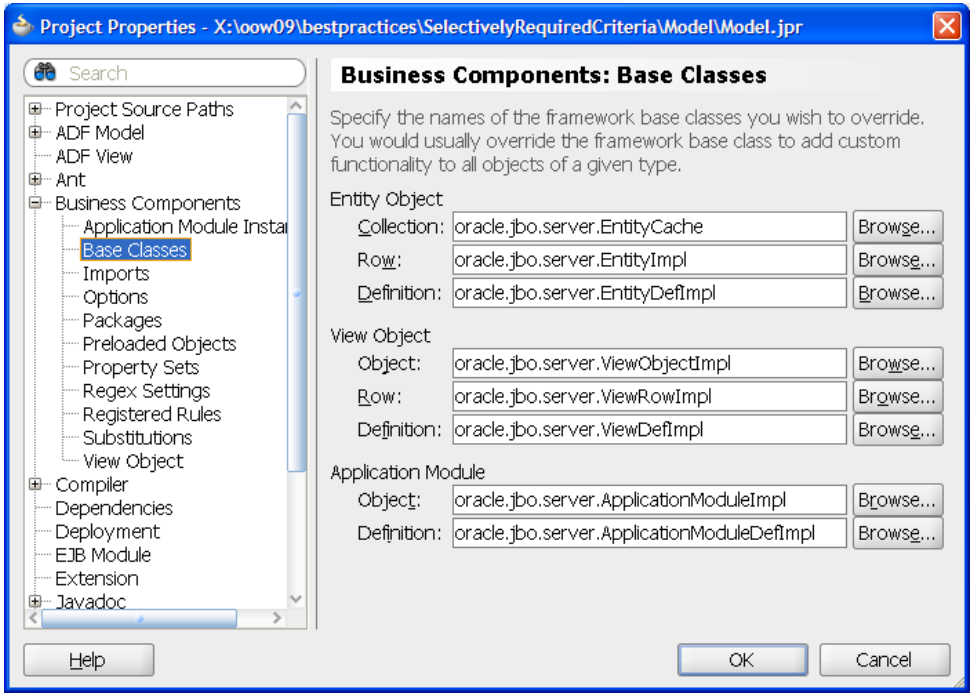
Architecture



ADF Business Components

Use a Layer of Framework Extension Classes





General Considerations

Custom Framework Extension Classes

- Always create and use custom base-classes for at least
  - ApplicationModuleImpl
  - EntiyImpl
  - ViewRowImpl
  - ViewObjectImpl
- Note that creating custom super classes for <object>Def classes are less common

Entity & View Object

When to generate Java Impl files

- Generate Java Impl files when entity objects are accessed from Java or Groovy
- Ensure code that accesses entity objects directly uses type safe APIs instead of the generic framework APIs
- Do not generate Impl classes without a need
  - Keep code base size reasonable.
  - Impl classes for the *def classes are rarely needed

View Criteria

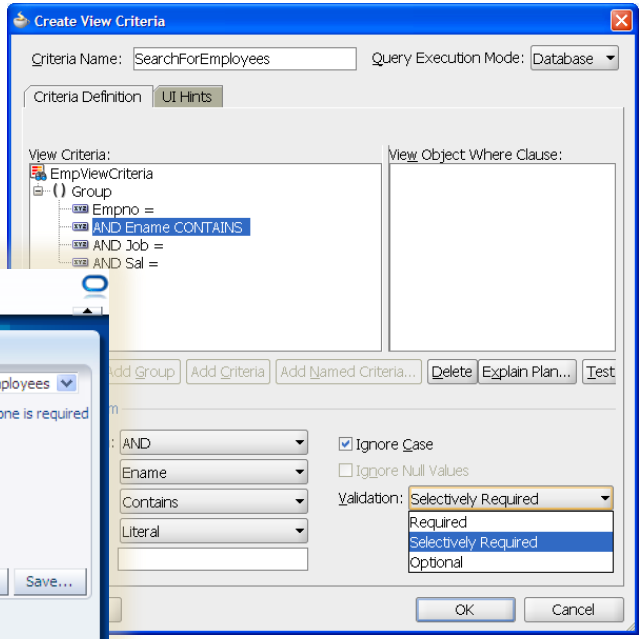
Prefer One View Object with Many View Criteria

- Not necessary to create separate VO that only differ by their WHERE clause
- Apply view criteria declaratively when defining view accessor and AM data model instances
  - Use bind variables with View Criteria
  - Be aware that "Ignore Null Values" options is of bad performance
- Avoid direct user input in ADF BC queries to protect against SQL injection attacks

View Criteria

Selectively Required Option

- Prevent user from doing "blind" query
- Typically includes your indexed columns

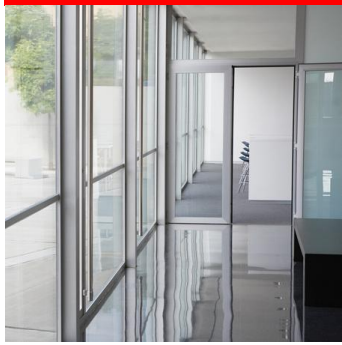


General Considerations

Think Passivation

- Passivation occurs when the application module used by a user is passed on to another user request
- Keep in mind that
  - AM should be tuned to not passivate too often
  - Java objects are not passivated
    - Consider the use of transient attributes and mark them for passivation
    - Alternative option: Use UserData Map in ADF BC to save information you need for longer

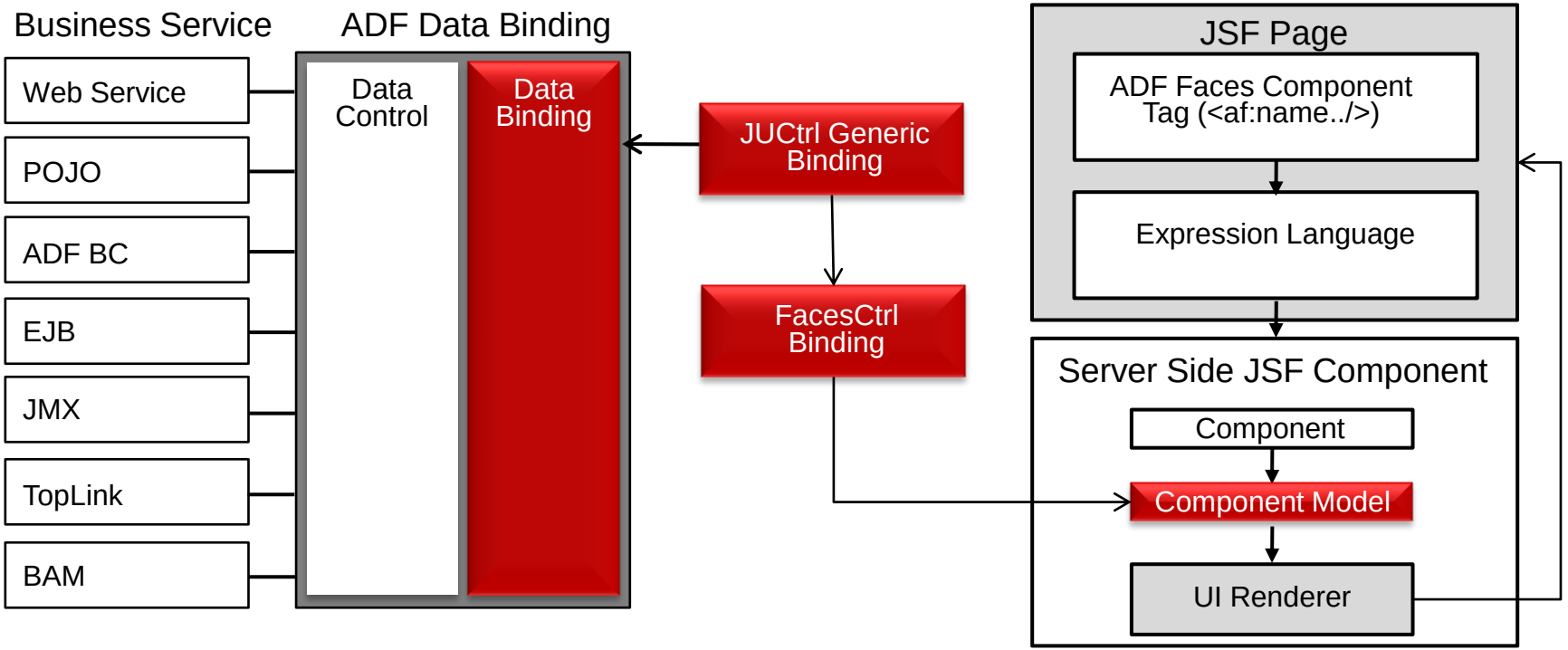
Program Agenda



- ADF Business Components
- ADF Binding Layer
- ADF Controller
- ADF Faces
- JavaScript
- WORA

Understanding ADF Bindings

How Oracle ADF binds into JSF



Programming Against the Binding Layer

- Expression Language
  - Current binding layer can be accessed from #{bindings}
  - Use from JSF components
  - Use from PageDef file
- Java
  - ADF binding layer is represented by Java objects at runtime
  - **`BindingContext.getCurrent()`** is the generic entry point
  - Binding can be accessed from UI components (for example: af:tree -> ((CollectionModel) getValue()).getWrappedData()

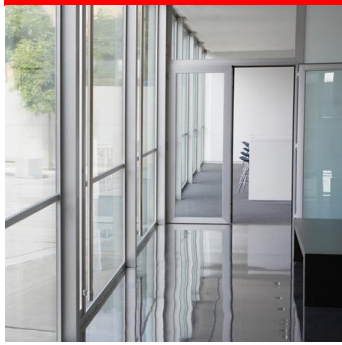
Typical Programming Mistakes

- Accessing the binding context from #{data}
  - Legacy code
- Accessing the binding layer outside of JSF and ADF request cycle
  - Filter in web.xml -> too early in the request cycle
  - Task flow initializer / finalizer -> no associated PageDef file
  - PhaseListener -> before RESTORE_VIEW causes NPE
- "Pinning" binding reference
  - Saving bindings in a property of a managed bean that is in a scope longer than request
  - Managed bean outlives binding container refresh cycle -> stale data

Typical Programming Mistakes

- Attempt to access binding that is out of scope
  - Bounded task flow in region cannot access binding container of parent view
- Call BindingContext.getCurrent() to access ApplicationModule (AM Impl) to call method on it
  - Bypasses binding layer
  - Bypasses ADF error handling
- Release binding container
  - ADF framework handles binding container and iterator lifecycle. No need to explicitly release a binding
  - Cause: Legacy ADF BC coding rules
- Fighting the framework

Program Agenda



- ADF Business Components
- ADF Binding Layer
- ADF Controller
- ADF Faces
- JavaScript

Task Flow Oriented Design

- Think task flow from design time on
- Think "unit-of-work" and factor task flow functionality into subflows
  - Share Data Control frame with sub-flows
  - Hide sub-flows from showing in ADF library
  - If you cannot fully explain and describe a task flow in 60 seconds it probably is too big

Use Task Flow as a "Common Language"

- Use task flow diagrammer as "common language"
  - Use task flow templates as blue prints
  - Use verbs and meaningful names in control flow case names
    - editOrder, createOrder, manageEmployee
- Use the display and description properties of the contained activities to explain the non-obvious

Task Flow Design

Task Flow Sizing Considerations

- Small task flows:
  - Require a lot of calling and maintenance management overhead
  - Reasonable sizes for distributed team development
  - Provide ultimate flexibility in architecture
- Large task flows:
  - Require less calls and maintenance management overhead
  - Less flexible as you can't call discrete functionality within the flow
  - Memory footprint likely bigger than small task flows

Task Flow Design

Task Flow Sizing Considerations

- Meet in the Middle
  - Larger task flows built out of several smaller task flows
  - Only top level task flow exposed for reuse
  - Encapsulates task flow dependencies
    - Good for distributing work among developers
    - Dependency management "by agreement" in smaller teams
    - Smaller memory foot print through load-on-demand and task flow private memory scopes

Best Practices

- Use input parameters and return values as the only API
- Document task flows with diagrammer notes
- Save task flow input parameter values in managed beans in pageFlowScope
  - Easy to document
  - Type safe API
  - EL accessible
  - Easy to discover and manipulate task flow input values
- Always use the smallest possible scope for managed beans
- Don't reference managed beans in session or application scope

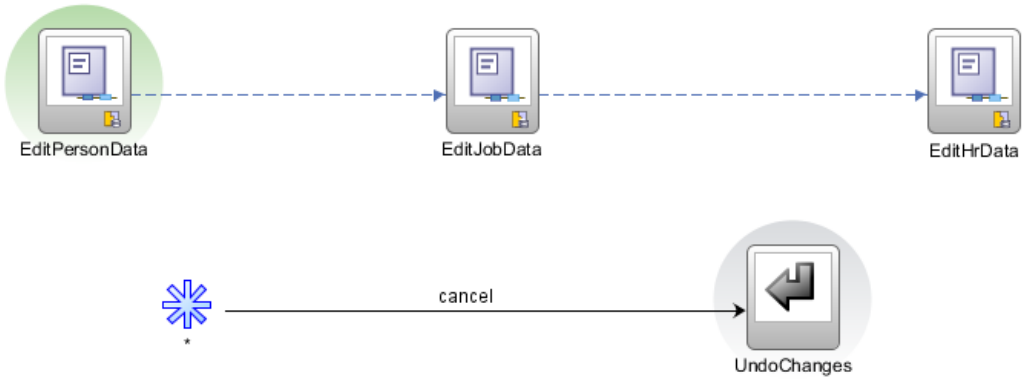
Best Practices

- Use shared data control scope as often as possible
  - Less database connections
  - Automatic parent view collection to child task flow detail collection synchronization
- Define an exception handler activity in every bounded task flow
- Make use of Task Flow Templates
  - e.g. to define exception handlers for all bounded task flows

Best Practices

Wildcard Navigation

- Avoid cluttering diagrams for common navigation rules
- Use wild card navigation for common functionality
  - Cancel
  - Commit



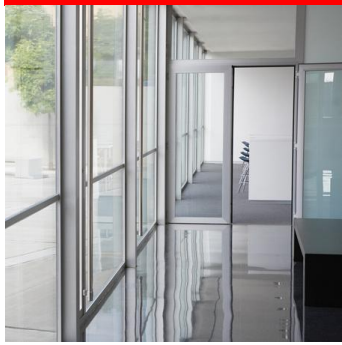
Typical Programming Mistakes

- Trying to access parent view managed bean objects or bindings
  - Use object references in task flow input parameters instead
- Use managed bean in session scope to share data
  - Consider shared data control or input parameters instead
- Use managed bean in request or backing bean scope for bean that switches task flows in dynamic regions
  - Use at least view scope to avoid ADF binding conflicts

Typical Programming Mistakes

- Use of `NavigationHandler().handleNavigation()` to navigate in bounded task flows
  - Queue action on hidden command button
  - queueActionEventInRegion on af:region

Program Agenda



- ADF Business Components
- ADF Binding Layer
- ADF Controller
- ADF Faces
- JavaScript

Managed Beans

- Encapsulate all model manipulation code in custom AM and/or VO methods & invoke them declaratively
  - Makes your app easier to regression test
- Only code that belongs in a JSF managed bean is…
  - Complex, dynamic UI component manipulation
  - Complex conditional navigation
- Even if you require backing bean code, invoke custom AM or VO methods using action binding
  - Guarantees uniform error handling

Managed Beans

Base Classes

- Managed beans can extend a base class
  - Super class can provide common functionality as public methods
    - e.g. ValueExpression, MethodExpression utility methods
  - Comparable to JSFUtil and ADFUtil but EL accessible
    - Utility classes are typically static classes

Managed Beans

Backing Bean

- Backing beans are managed beans with a tight association to a view
  - Contains UI component references
  - Usually not used with more than a single view
  - Must be in request or backing bean scope
- Backing beans are Java objects and as such
  - Increase the application code base to maintain, migrate and further develop
- Recommendation
  - Create backing beans only when the use case requires it

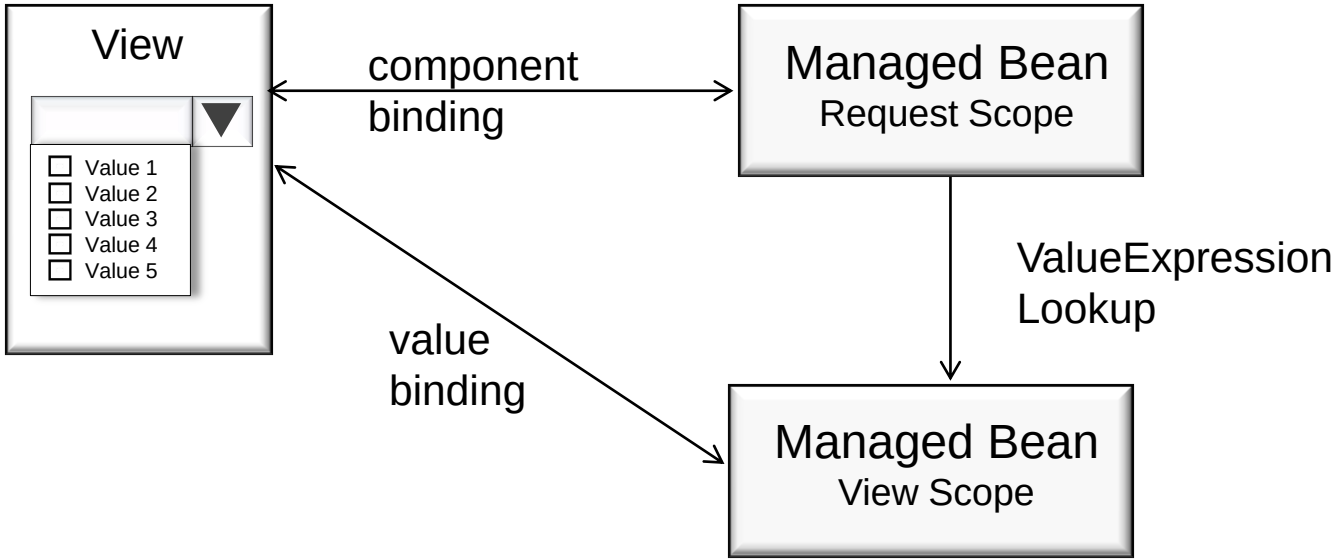
UI Manager Pattern

Two Managed Bean Problems – One Cause

- Problem #1 – JSF component serialization exception
  - JSF components are not serializable.
  - Any attempt to bind JSF components to a bean in a scope broader than request fails with an exception thrown
- Problem #2 – JSF component values are not persisted in managed bean
  - Persisting component value states in managed beans requires scopes broader than request
  - Values in request scope reset after each request leading to loss of user data entry and failed value change listener invocations

UI Manager Pattern

Two Managed Bean Problems – One Solution



Typical Programming Mistakes

Mixing HTML and JSF components

- HTML and JavaServer Faces are different technologies
  - Markup oriented vs. component oriented
  - Compile – Render, vs. multi stop request lifecycle
- Facelets in JSF 2.x support mixing of HTML and JavaServer Faces components
  - Prior to Facelets and JSF 2.x, mixing HTML and JSF is a no-go
  - Still with JSF 2.0 problems may occur because ADF Faces components come with their own sizing behavior (geometry management)
  - Adding HTML to the body of a JSF or ADF Faces component may break component functionality
- Recommendation: Build pages and views with ADF Faces components

Typical Programming Mistakes

Use of inlineStyle and contentStyle CSS

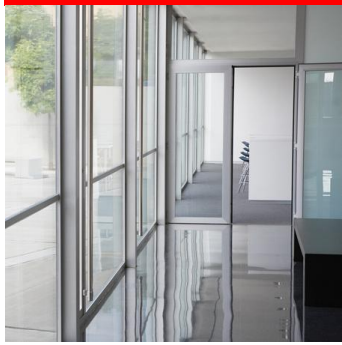
- inlineStyle property allows developers to add CSS to a component rendering
  - CSS is directly added to the page output
  - Recommendation: Use ADF Faces skinning and reduce the use of CSS to the absolute minimum (e.g. conditional color coding)
- contentStyle property allows developers to add CSS to the value content area of an ADF Faces component
  - Styles component content area better than inlineStyle but has same issued as mentioned above

Typical Programming Mistakes

ui:include and jsp:include

- ADF PageDef file is not aware of ui:include or jsp:include references in a view
  - Included fragments fail at runtime if they use ADF bound components
  - You can copy ADF bindings used by the included content into the view's PageDef file.
    - Prevents reuse of the fragments
- Recommendation
  - Use ui:include and jps:include for layouts only (if at all)
  - Use ADF regions to add ADF bound content to a page at runtime

Program Agenda



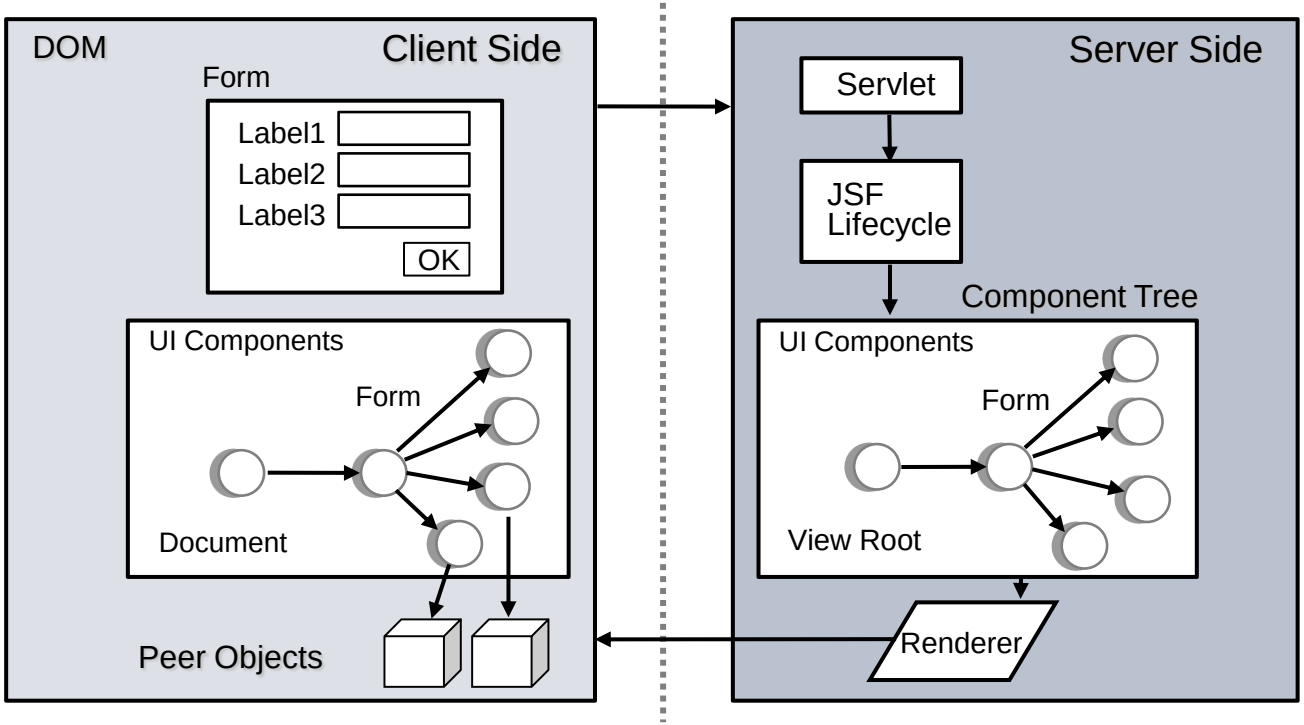
- ADF Business Components
- ADF Binding Layer
- ADF Controller
- ADF Faces
- JavaScript

You can use JavaScript …

If following these rules!

1. Know what you do and why you do it
2. Understand the ADF Faces client side JavaScript framework
3. Use JavaScript by exception

ADF Faces Client JavaScript Architecture



ADF Faces Client JavaScript Architecture

Features

- JavaScript component API
- af:clientListener
  - Listens to DOM and ADF Faces component events
- af:serverListener
  - XMLHttp JavaScript call to server
  - Queued as custom event on client
  - Answered by managed bean
- `ExtendedRenderKitService`
  - Trinidad class to invoke JavaScript from Java

ADF Faces Client JavaScript Architecture

Benefits

- Component oriented API
- Integrated with JavaServer Faces and ADF Faces request lifecycle
- Ensures JavaScript calls to work on all ADF Faces certified browsers
- Ability to listen to component events like query, selection, popup launch etc.
- Allows you to suppress component events from propagating to the server
- Easier to learn and deal with than vanilla JavaScript programming

Typical JavaScript Programmer Mistakes

- Access the browser DOM tree when coding with JavaScript in ADF Faces
  - DOM tree does work against generated HTML and not against components
- Use or manipulate objects ending with "Peer"
  - Use ADF Faces public APIs only: AdfRich<component name>
- Use of methods that are all upper case or camel case with a uppercase first letter
  - These methods are for internal use only
- Use of objects in JS packages that have the name "internal" in them

Typical JavaScript Programmer Mistakes

- Think that using JavaScript saves you a round trip
  - JSF to work properly requires the server side component state to be synchronized with the client
  - Suppress server round trips only if no component state has been changed on the client

ORACLE

Q&A

twitter.com/fnimphiu


ORACLE®