ENTERPRISE DEVELOPMENT SERVICES DIRECT FROM THE CREATORS OF SPRING FRAMEWORK

INTERFACE21

Spring Web Flow:

Enabling High Level, Low Effort Web Conversations

Colin Sampaleanu
Interface21

Spring

About Me

- Spring Framework core developer since mid-2003
- Founder and Principal Consultant at Interface21, a unique consultancy devoted to Spring Framework and Java EE
  - Training, consulting and support
  - “From the Source”
  - http://www.interface21.com

Java Web Frameworks: Today's Landscape

- Container model: Servlets, Portlets
- *Request-oriented Web MVC Frameworks*
  - Struts, *Spring Web MVC, W*WebWork 2, etc.
- Event-driven, component-oriented Web MVC Frameworks
  - Tapestry, JSF, Wicket, Echo, etc.
- Java only in the back-end:
  - GWT, Tibco GI, Backbase, Flex

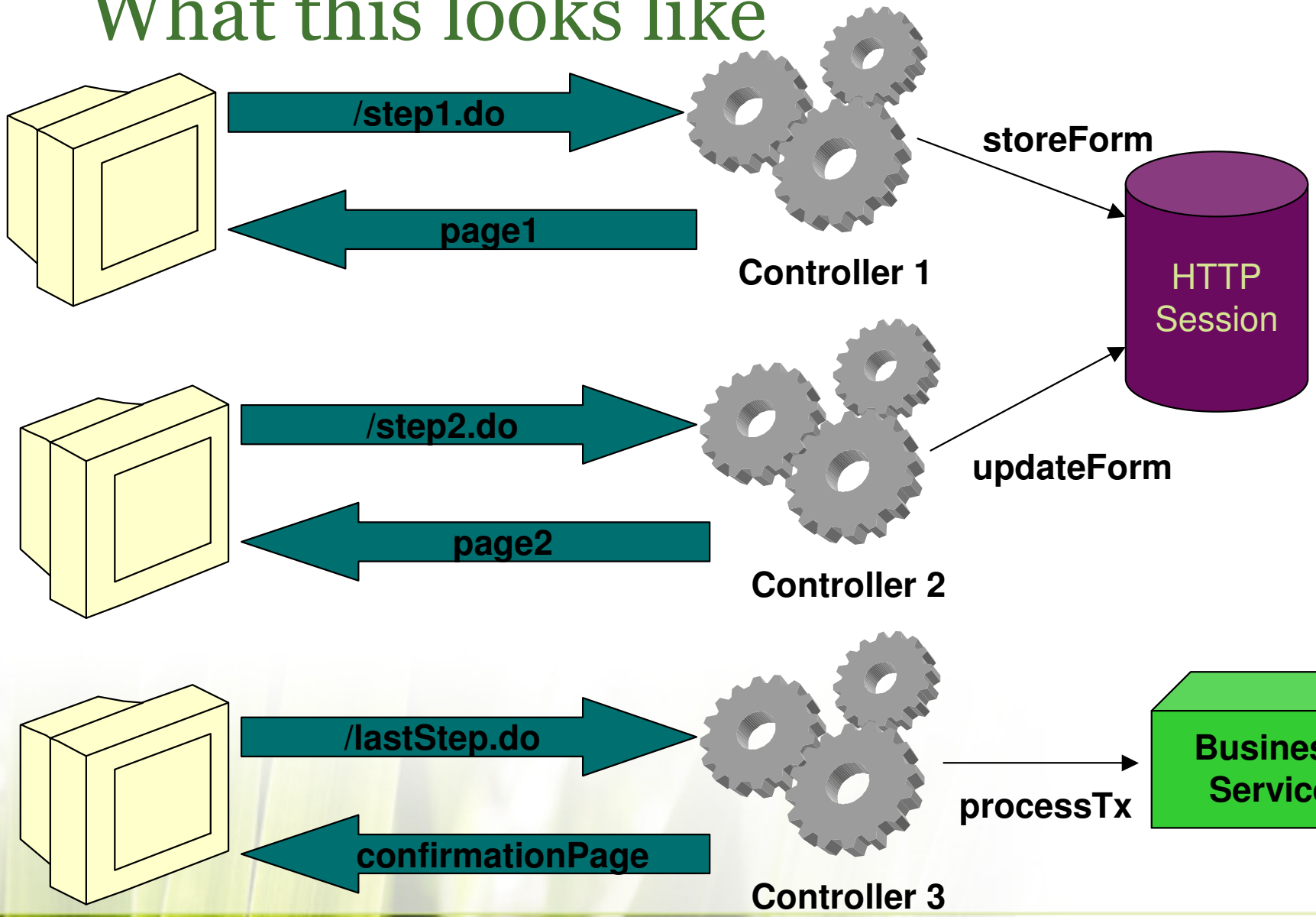
Page Flows in Web Apps: Where's the need?

- By example
- Take a simple use case:
  - A wizard for a phone operator to use to sell items to customers
- Characteristics:
  - An "application transaction" that spans several steps
  - Some steps solicit user input
  - Some steps are decision points
  - Some steps perform calculations
  - Navigation from step-to-step is controlled

How would you do this today with Struts?

1. Create a session-scoped ActionForm to hold the wizard form data
2. Define a JSP for each step
3. Define an Action for each step
4. Expose each Action at a request URL
5. Have the form rendered by the JSP submit to that URL
6. At the end, delegate to a business service to commit the transaction

What this looks like



Issues with this approach

- Request centric: no concept of an ongoing conversation or flow
- Brittle dependency on request URLs
- Manual state management
- Odd new window behavior
- Proper back button behavior is difficult
- “Loose” controlled navigation
- Difficult to observe process lifecycle
- Controller and command logic are coupled
- Heavily tied to HTTP

What About Higher Level Frameworks Like JSF? Is it any Better?

1. Establish an object to hold the wizard form data. This may be an existing domain object, or a new JSF backing bean for the purpose.
2. Define a JSP view for each step
3. Some steps can be handled with simple declarative JSF navigation, because the transitions are very simple an unambiguous.
4. Some steps require Java action handling code in the backing bean, to make complex navigation decisions. Java code returns a string result, then JSF navigation rules map this to a view.

Built-in Page Navigation in JSF

- JSF allows simple navigation rules to be defined
- From `faces-config.xml`:

```
<navigation-rule>
  <from-view-id>/carDetail.jsp</from-view-id>
  <navigation-case>
    <description>
      Any action that returns "confirmChoices" on carDetail.jsp
      should cause navigation to confirmChoices.jsp
    </description>
    <from-outcome>confirmChoices</from-outcome>
    <to-view-id>/confirmChoices.jsp</to-view-id>
  </navigation-case>
</navigation-rule>

<navigation-rule>
  <from-view-id>/storeFront.jsp</from-view-id>
  <navigation-case>
  ...
```

Issues with this approach

- We're working a bit higher level than Struts, but not much
  - Request centric: no concept of an ongoing conversation or flow
  - Components handle state, but manual state management as far as any multi-page flow is concerned. State is tied to Session or Request, not the conversation
  - Proper back button behavior is difficult
  - "Loose" controlled navigation
  - Difficult to observe process lifecycle

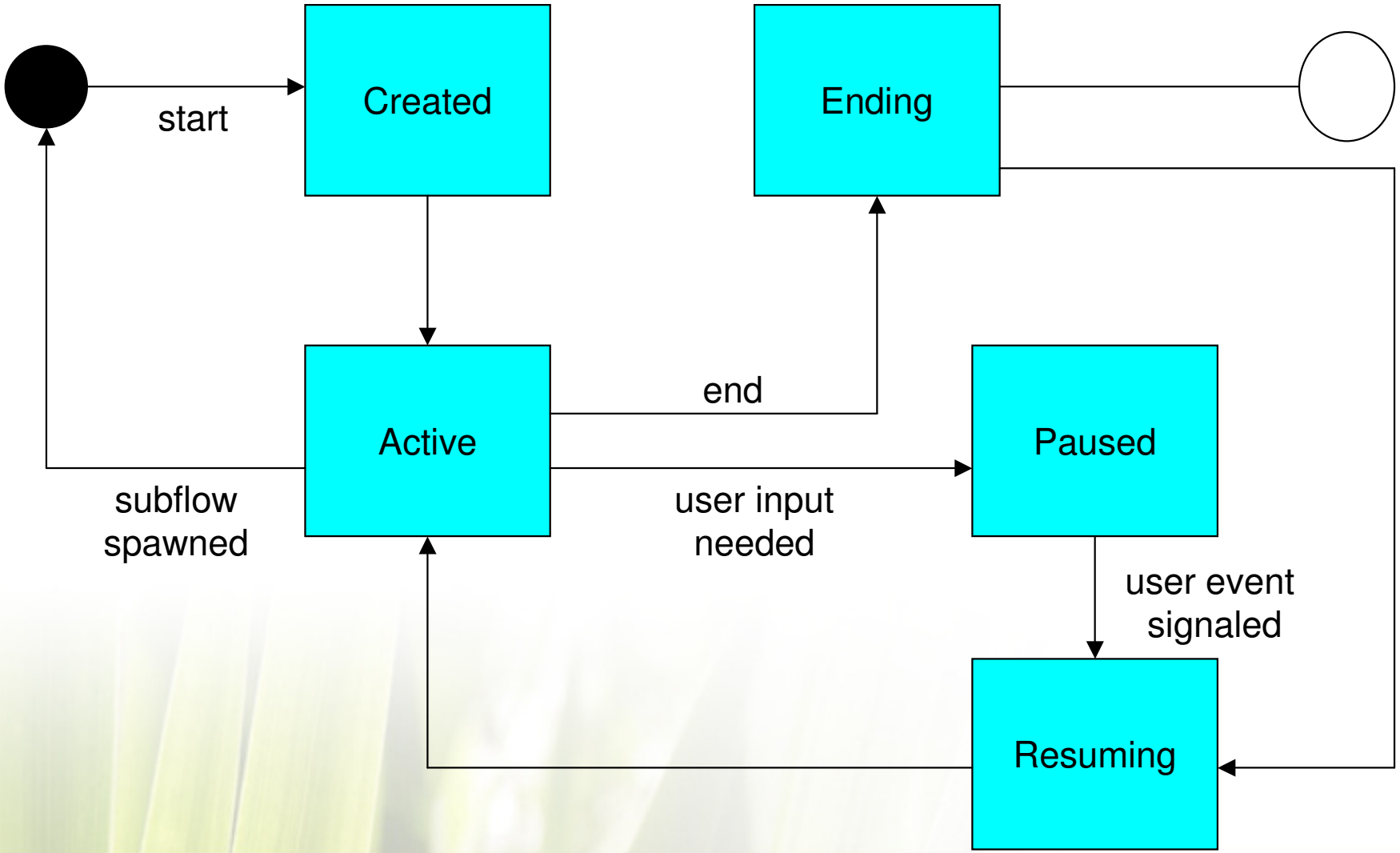
Consequences

- **Many lines of custom code are written to address these issues**
- As an application grows more complex maintainability breaks down
- Fundamentally, something is missing for certain types of web apps
- Traditional approaches today, including the raw JSF navigation mechanism lack a key abstraction: **the Flow**
- A Flow is typically longer than a single request but shorter than a session
- It is a conversation. It's Spring Web Flow!

Significant architectural differences

- One flow drives the entire conversation
- When user input is required, the flow “pauses” and control returns to the client
- Clients “signal” events to tell the flow what happened
- The flow responds to events to decide **what to do next**
- What to do next is fully encapsulated
  - Flows are modular “black boxes”

Flow Execution State Transition Diagram



Question

Q: How do you program the Flow?

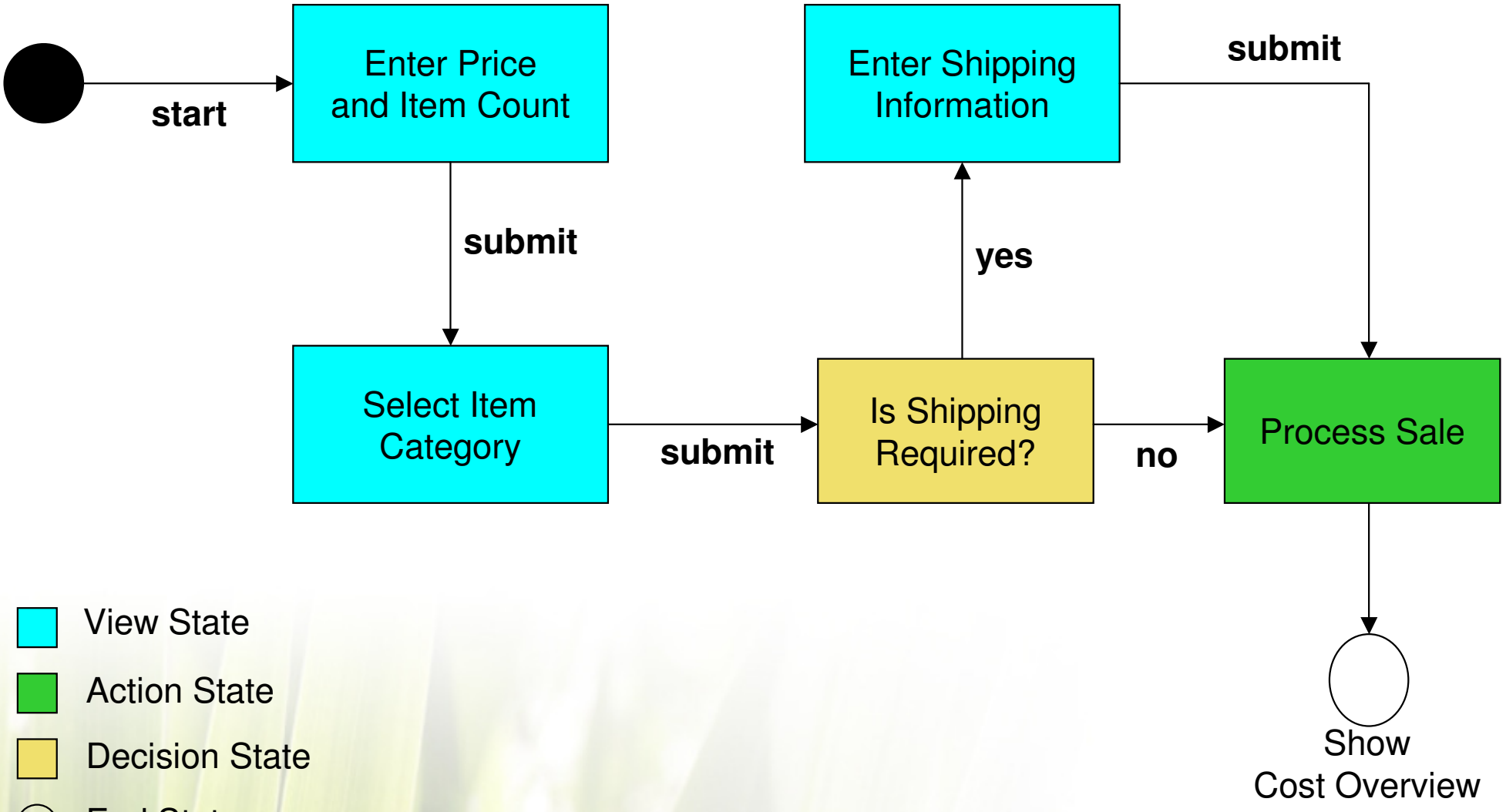
Q: How does it know what to do in response to user events?

A: You create a Flow definition

Flow Definition Structure

- A Flow definition serves as instructions to a finite state machine
- It consists of a set of states that you define
- Each state executes a polymorphic behavior when entered
  - View states solicit user input
  - Action states execute commands
  - Subflow states spawn child flows
  - End states terminate flows
- Events you define drive state transitions

The "Sell Item" Flow Definition



Code Break!

- The “Sell Item” Flow Definition
  - If viewing on-line, see presentation notes
- Demo of the Flow Execution

Some Big Advantages Emerge

- Navigation enforcement
- Modularity & Decoupling
- Automatic state management
- Abstraction
- Facilites Agile Development Cycle
- Flow lifecycle visibility
- Toolability

Navigation Enforcement

- Web Flow ensures defined navigation rules can not be bypassed or short-circuited
  - This happens for free
  - Without losing any power compared to other approaches

Modularity & Decoupling

- Navigation rules are encapsulated in one place
  - Controller logic is clearly decoupled from command logic
  - The Flow is the controller, deciding what state to enter next
  - States execute arbitrary behavior when entered
- The logical flow is clearly defined, readable
- Flows are Modules the same way a class is a module

Automatic State Management

- Flow scope state is managed for you automatically
  - Conceptually no different than existing (Session, Request, etc. scopes)
  - Put something in flow scope
    - It's there while the flow lives
    - It's cleaned up after the flow is done

Abstraction

- Flows are very similar in different environments (JSF, Spring MVC, Struts, etc.)
  - A flow is very much a black box
  - Could be completely identical for different environments
  - Or slight differences where Web Flow capabilities overlap with base Web UI Framework capabilities
    - E.g. use Web flow validation, or JSF validation?

Facilitates Agile Development Cycle

- Flows are re-compiled automatically, on the fly
  - Test your changes without restarting servlet container

Toolability

- Flow definitions are extremely toolable
- Well suited to graphical editing
- Well suited to programmatic generation
- Spring IDE 2.0 knows about Web Flow
  - Web Flow Graphical Editor
  - Web Flow XML Editor
  - Use for visualization: a communication tool
  - Use for editing: easier for new developers

Toolability

- Spring IDE Demo…

Flow Lifecycle Visibility

- It is easy to attach observers to the flow lifecycle
  - Do things at certain points in the flow
  - Extend basic Web Flow capabilities

Navigation in Web Apps: The Reality

- Simple web apps may get by with free form navigation

- For most web apps, we need a combination:
- Free form navigation
  - Controlled navigation, ideally managed in a high-level fashion with something like Spring Web Flow

Design Tips

- What makes a good Web Flow?
  - Accomplishes some business goal, e.g.
    - *Book a trip*
    - *Pay your taxes*
    - *Apply for a Loan*
  - Encapsulates that business goal as a reusable module
    - A “black box”
  - Entry point to business goal is a public URL, but intermediate steps are valid only in the context of the flow (i.e. may not be bookmarked)
  - Often has multiple paths to support different scenarios
    - Paths may vary based on contextual information
  - Often spawns other Flows as *sub flows*
    - Has a clear, observable lifecycle: Flow executions start, events happens, they end

Design Tips

- What doesn't make a good Flow?
  - Any set of pages with free-form navigation between them
    - Publicly accessible URLs
    - Often able to be bookmarked
    - Often backed by individual controller or controller chain
  - E.g. Index pages, welcome pages, item detail pages, status pages, menus, simple form submissions (mailers, etc.)
- Web flows are best suited for enforcing controlled navigation to support business process workflows
  - As a compliment to traditional controllers
  - Don't abuse them

Web Flow Integration with Lower Level Web UIs

- Spring Web Flow works mostly as a black box.
  - An event is received
  - One or more flow states are traversed
  - When a view state is reached, control returns to the underlying UI framework, which displays the view (typically in a browser)
- Web Flow is not coupled with any other web framework
  - Not even Spring MVC or the Servlet API
  - Proven in a Portlet environment
  - Truly an embeddable component

Web Flow Integration: In General

- Create a specialized controller for that environment, as a ‘bridge’
  - For example:
    - The FlowController for Spring MVC
    - The FlowAction for Struts
    - The FlowNavigationHandler and FlowPhaseListener for JSF
    - The PortletFlowController for Portlets

Web Flow Integration (3)

- Of course, some integrations are easier than others:
  - Struts and Spring MVC integration is trivial, because for those frameworks, Web Flow can handle all state management, binding, and validation
    - Here, Web Flow binding/validation == Spring MVC binding/validation
  - JSF is tougher fit
    - Higher level
    - Stateful components
    - Basic validation
    - Complex lifecycle

JSF – Web Flow Integration

- We arrive at an integration that tries to leverage as much as possible from each framework
  - Web Flow assumes all navigation duties
  - Stateful JSF pages/components
  - JSF binding for component state to model data
  - Simple JSF validation for individual fields
  - Optional use of Spring *Validators* for complex inter-field validation

What About Other Integrations?

- We may do more, as driven by user demand and resourcing
- Easy for third parties to do as well

- Required work is comparable to JSF, Struts, or Spring MVC integration
  - Web Flow's clean design ensures integration is possible with any similar framework

Key Concepts

The FlowDefiniton

- Definition of one or more states making up the flow
- One start state
- Some intermediate states
- One or more end states
- States have transitions between each other based on events

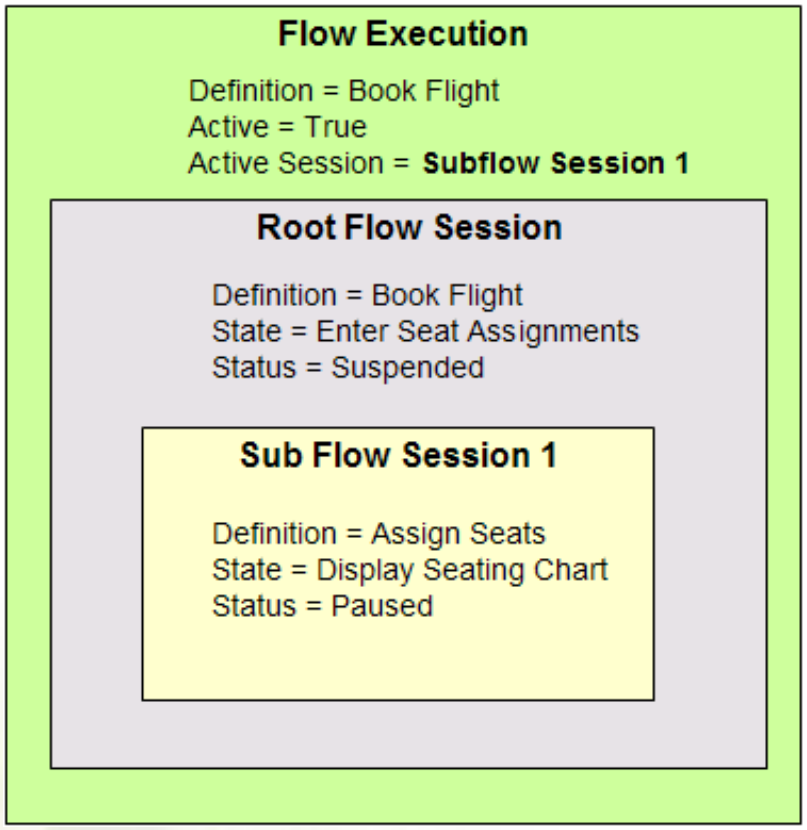
- May be built via Java API (see Phonebook sample)
- More common to define via metadata such as XML
- Internally, *XMLFlowBuilder* parses the latter to produce a FlowDefinition

FlowSession

- Runtime instantiation of a FlowDefinition
  - A FlowDefinition is to a class what a FlowSession is to an object instance of that class

FlowExecution

- Since Flows nest, FlowSessions live inside FlowExecutions



States – 5 Main types

- View State
  - Pause and allow the user to participate in a flow
- Action State
  - Execute business logic, usually indirectly
- Decision State
  - Make a flow routing decision
- Subflow State
  - Spawn a sub flow
- End State
  - Terminate a flow

State Transitions, Including Global

```
<xxx-state id="state1">
    <transition on="localEvent1" to="state2"/>
</xxx-state>

<xxx-state id="state2">
    <transition on="localEvent1" to="state1"/>
</xxx-state>

<global-transitions>
    <transition on="globalEvent1" to="state1"/>
    <transition on="globalEvent2" to="state2"/>
</global-transitions>
```

Actions

- Directly invoked by ActionState
- Also possible to invoke on:
  - Flow start
  - State entry
  - On transition
  - On state exit
  - Before view rendering
  - On flow end

Different Types of Actions

- Standard
  - Must extend ActionState
- Multi
  - Must extend MultiAction
- Bean / Decision Bean
  - Your POJO is simply wrapped and called
- Evaluate Action
  - Evaluates an expression you specify inline
- Set Action
  - Set an attribute in flow or other scope

Example Decision Bean Action

```
<action-state id="shippingRequired">
    <bean-action bean="shippingService"
                 method="isShippingRequired">
        <method-arguments>
            <argument expression="${flowScope.purchase}"/>
        </method-arguments>
    </bean-action>
    <transition on="yes" to="enterShippingDetails"/>
    <transition on="no" to="placeOrder"/>
</action-state>
```

Example Evaluate Action

```
<action-state id="getNextInterviewQuestion">
    <evaluate-action
    expression="flowScope.interview.nextQuestion()"/>
        <evaluation-result name="question"/>
    </evaluate-action>
    <transition on="success" to="displayQuestion"/>
</action-state>
```

FlowExecution Repositories

- Storage for the flow runtime data
- Strong relationship to how many instances of the FlowExecution you can actually have. Affects things such as back button behavior
  - Simple
  - Continuation
  - Client Continuation
  - Pluggable: expect extensions

New Scopes

- Flow Scope
  - Depending on FlowExecution repository type, there may be multiple copies of this scope
- Conversation Scope
  - May be considered global to the flow execution. Attributes put in here will be retained for the life of the flow execution and will be shared by all flow sessions.
- Flash Scope
  - Attributes here are preserved until the next user event is signaled into the flow execution. Useful for redirects.

Launching Flows

- Parameter style anchor

```
<a href="flowController.htm?_flowId=myflow">Launch My Flow</a>
```

- Via form:

```
<form action="flowController.htm" method="post">
   <input type="submit" value="Launch My Flow"/>
   <input type="hidden" name="_flowId" value="myflow"> </form>
```

- REST-style anchor:

```
<a href="flowController/myflow"/>Launch My Flow</a>
```

- JSF, via commandLink:

```
<h:commandLink value="Go" action="flowId:myflow"/>
```

Resuming Flows

- Resuming via anchor:

```
<a href="flowController.htm?_flowExecutionKey=
   ${flowExecutionKey}&_eventId=submit"> Submit </a>
```

- Resuming via form (just one variant):

```
<form action="flowController.htm" method="post"> ...
   <input type="hidden" name="_flowExecutionKey"
       value="${flowExecutionKey}">
    <input type="hidden" name="_eventId" value="submit"/>
    <input type="submit" class="button" value="Submit">
   </form>
```

Attribute Mappers

- Map attributes going into or out of a flow
  - Input mapper: maps attributes going into a flow as it starts
    - coming from the external source
    - from a parent flow (parent side and/or subflow side)
  - Output mapper: maps attributes leaving the flow as it terminates
    - Subflow side or parent side

```
<input-mapper>
  <mapping source="shipping" target="flowScope.shipping"/>
</input-mapper>
```

Post+Redirect+Get in Web Flow

- Eliminates common “double submit” issue
- Either explicit redirect via `redirect` prefix

```
<view-state id="displayList" view="redirect:yourList"> <transition
    on="add" to="addListItem"/> </view-state>
```

- or automatic redirect via `alwaysRedirectOnPause` attribute (default)

Other redirects

- Flow Definition Redirect (redirect to flow)
  - Requests a redirect that launches an entirely new flow execution
  - Used to support *flow chaining* and *restart flow* use cases
- External Redirect
  - Requests a redirect to an arbitrary external URL

Code Break!

- Let's look at how easy it is to test flows, and some other interesting details.

The Future

- Web Flow 1.0 is a strong foundation on which to build
- It will continue to evolve based on actual user needs
- It will continue to evolve and be usable for a greater range of tasks
  - Shorter-lived flows allowing use for free-form navigation
  - Become more general purpose for rich web applications (not web sites)
  - While maintaining strong integration with existing technologies like JSF

On The Road to 1.1...

- One-shot (request) Flows
- Flow Composition
  - The ability to compose flows together to form a graph / hierarchy
- Tighter UI integration
- More third-party integration
  - Securing Flows (Spring Security / Acegi)
  - Declarative Conversational Session Management (Hibernate)

Spring and AOP Training from the Source

Core Spring:

May 1st – 4th, Philadelphia, PA

Core AOP:

Simplifying Enterprise Applications with AOP

May 22nd – 25th, Philadelphia, PA

http://interface21.com/training

Spring

- Questions?