ENTERPRISE DEVELOPMENT SERVICES DIRECT FROM THE CREATORS OF SPRING FRAMEWORK

INTERFACE21

Spring Web Flow

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

Web Tier Evolution

- Initial approach is very low-level in the Wild West early days:
  - CGI with Perl/Python/TCL/C/C++
  - In-process CGI (PHP, mod_perl, mod_python, etc.)
  - Servlets: Java in-process CGI equivalent
- People get the job done, but the code is not very pretty

Web Tier Evolution (2)

- Separation of concerns begins
  - People realize that:
    - there is a need to separate developer roles
    - This is helped by seperating logic form presentation markup
    - This also allows better code reuse
- JSP Model 1
- Typical PHP approach

Web Tier Evolution (3)

- True separation of concerns happens: MVC frameworks become popular:
  - Request Driven Web MVC Frameworks
    - JSP Model 2
    - Struts, Spring MVC, WebWork, etc.
    - Ruby on Rails, Django, etc.
  - Event-driven, component oriented – closest to true MVC
    - Tapestry, JSF, Wicket, Echo2, etc.

MVC for the Web: What drives it?

- The business-tier contains the use cases
- The web-tier just provides a way to access those use cases
- Therefore:
  - A **thin** web-tier
    - Handling of request and rendering of responses: **nothing else**
  - A **clean** web-tier
    - Easy invocation of business logic
    - Business logic does not depend on the web-tier
    - Request processing decoupled from presentation markup

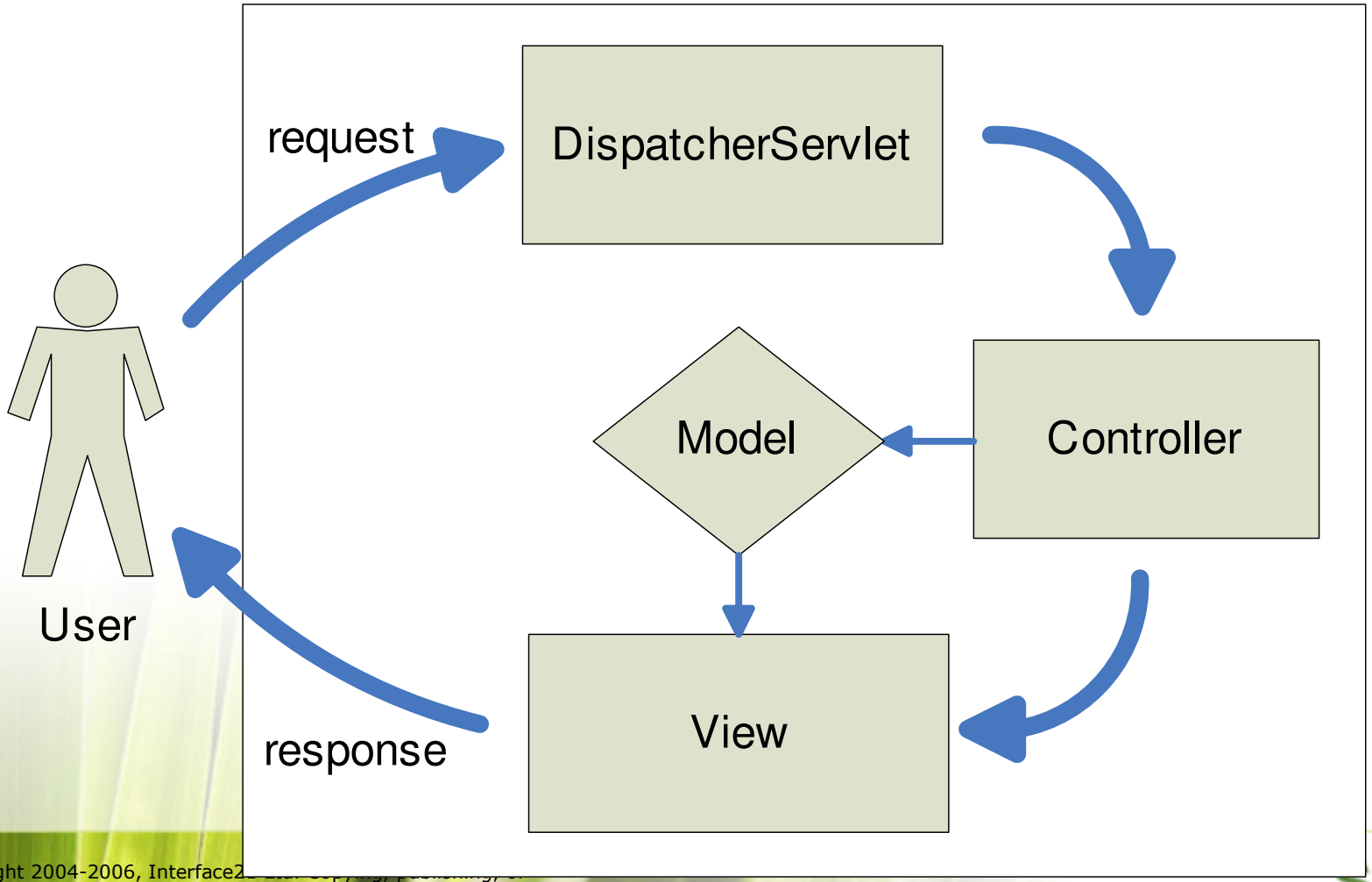
MVC for the Web: Core Concepts

- From these basic requirements we can identify three core roles:
  - A **Controller** (request handling)
    - Example: perform a search
  - A **Model** (data to render)
    - Example: the data returned by the search
  - A **View** (rendering)
    - Example: the search results put out as an HTML page or PDF file

Modern Java Web-App Frameworks

- *Request-driven Web MVC Frameworks*
  - Struts
  - *Spring Web MVC*
  - WebWork2
- Event-driven, component-oriented Web MVC Frameworks
  - Tapestry
  - Java Server Faces (JSF)
  - Wicket, Echo2, etc.
- Portlet Containers

Request Driven Frameworks: Spring MVC as an example



Event-Driven, Component-Oriented Frameworks

- Good request driven frameworks such as Spring MVC use a powerful model including pluggable controller and view resolution strategies, and strong data-binding/validation capabilities
- But working with a framework based on stateful components/pages, responding to events, is fundamentally a higher-level approach
  - (Nested) Components worry about markup
  - Framework worries about state for components
  - Closer feel to true MVC
- Still room for both: learning curve, constraints due to web environment, can be negatives

What JSF Provides

- Stateful components, which generate their own markup
  - JSP-centric templating implementations so far
  - Others coming: Facelets is very compelling
- Server-side handling of user-interface events
- Form handling and validation, with type conversion
- Very simple navigation, but pluggable
- Simple IOC container: e.g. inject business objects into UI objects
- Localization
- Usable but somewhat flawed extension model

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>
  ...
```

What About Tapestry?

- Very comparable to JSF in concepts, but generally a cleaner design
  - Less “design by committee” feel
- Not JCP standard, but popular (Apache)
- Very clean templates
  - True separation of roles
- Built on Hivemind IoC container
- Good Spring integration
- Better extensibility, except in navigation

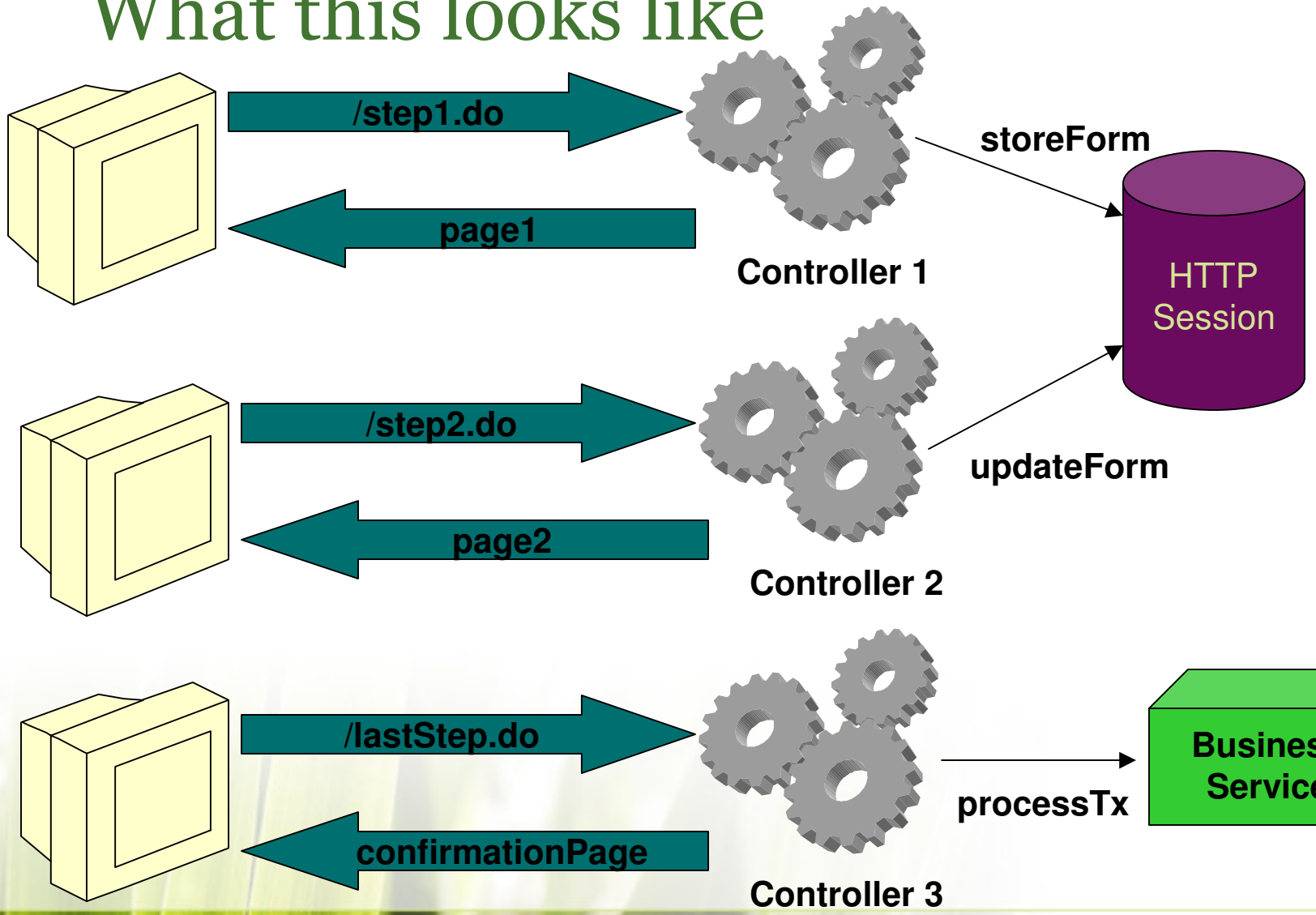
Page Flows in Web Apps

- 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

How Would You Do This With Raw JSF?

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.

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

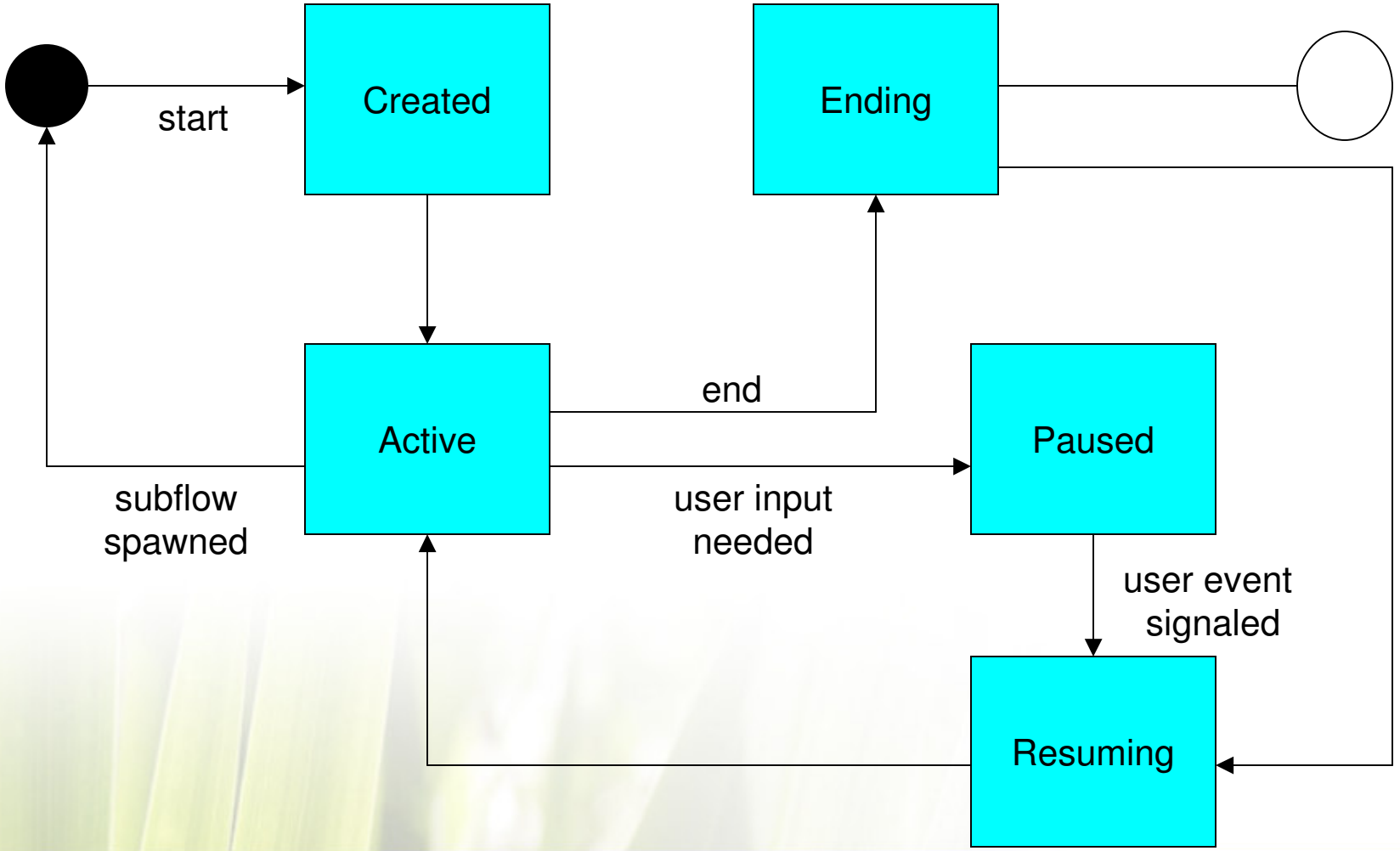
Consequences

- **Many lines of custom code are written to address these issues**
- As an application grows more complex maintainability breaks down
- Fundamentally, something is missing
- 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?

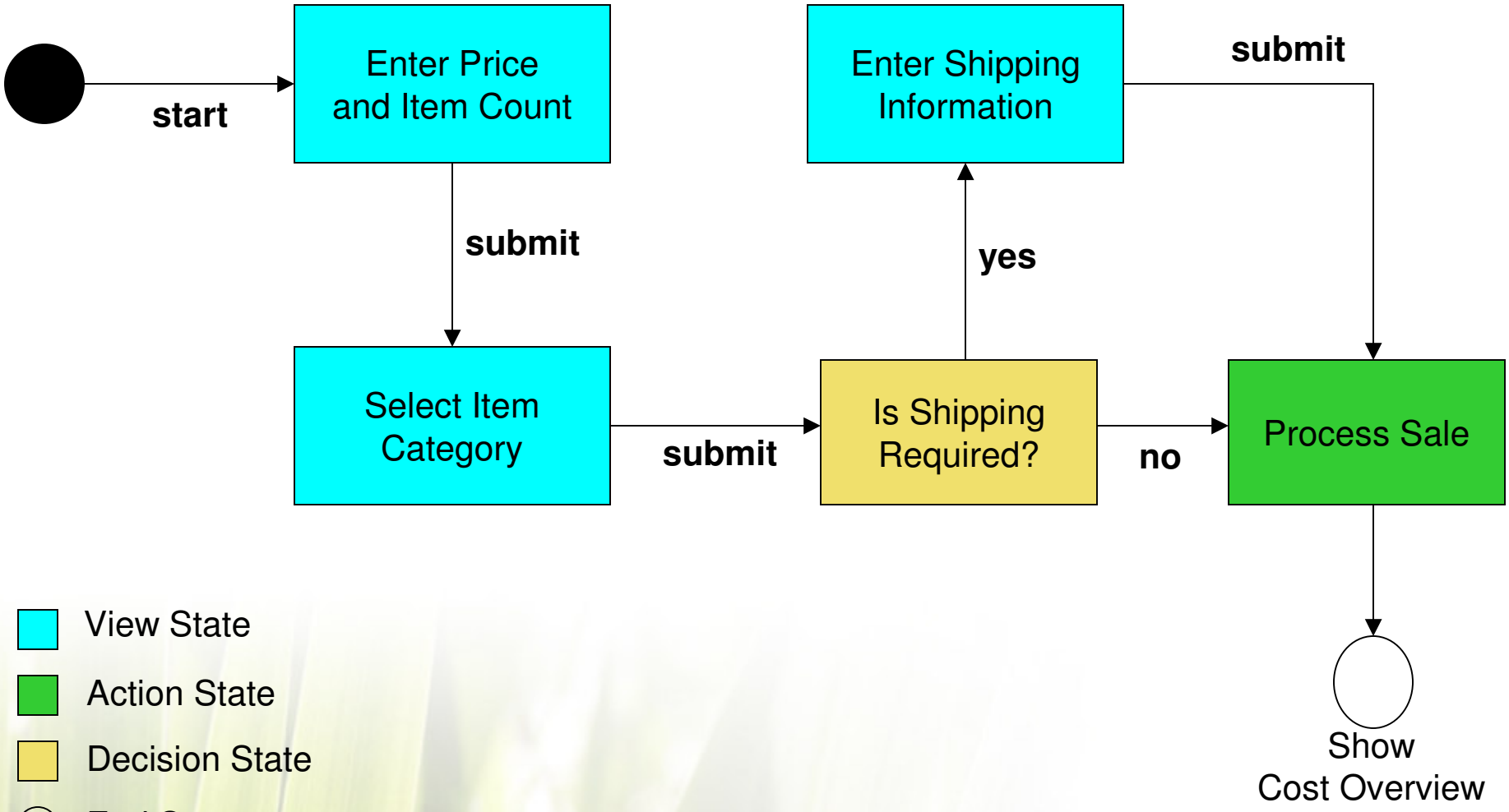
Q: How does it know what to do in response to user events?

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

Advantages

- The logical flow is clearly defined, readable
- Navigation rules are encapsulated in one place
- Navigation is strictly enforced
- State management is automatic
- Flow lifecycle is fully observable
- 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
- Flow definitions are extremely toolable

- Flows are very similar in different environments (JSF, Spring MVC, Struts, etc.)

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 Tapestry Integration?

- Important to us, but still to be done
- Required work is comparable to JSF integration, which has provided some valuable lessons
  - Lack of pluggable navigation concept in Tapestry, along with more complex state handling model in Tapestry makes the job harder
- However, Web Flow's clean design ensures integration is possible with any similar framework

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”
  - 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?
  - Index pages
  - Welcome Pages
  - Menus
  - Simple Form Submissions (Mailers, etc.)
  - Sites with a lot of free navigation
- Web flows are best suited for enforcing controlled navigation to support business process workflows
  - As a compliment to traditional controllers
  - Don't abuse them

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

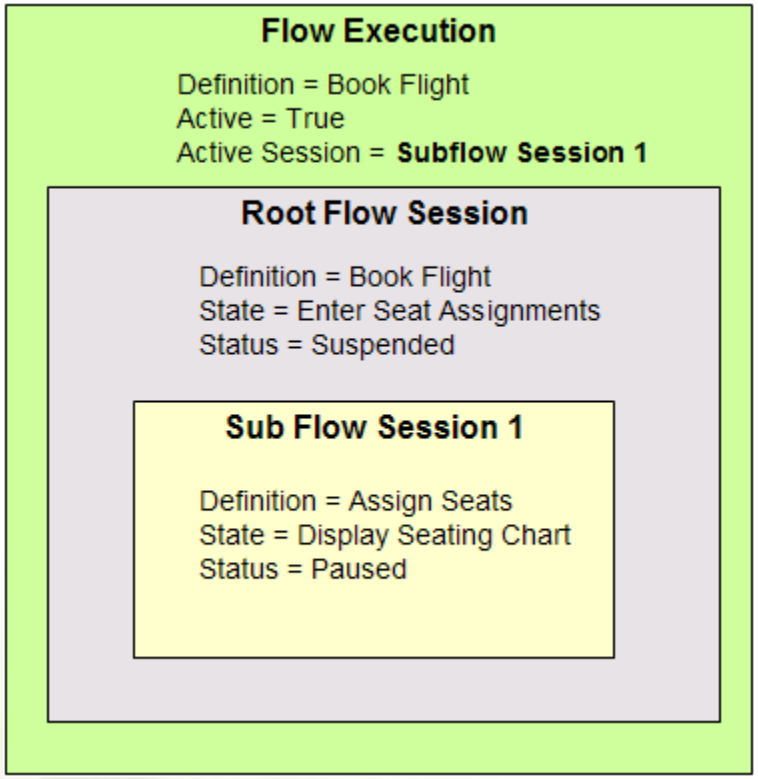
- May be built via Java API
- More common to define via metadata such as XML
- Internally, *XMLFlowBuilder* parses the latter to produce a FlowDefinition

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, and thus what the user can do in the flow
  - Simple
  - Continuation
  - Client Continuation

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.

Code Break!

- Let's look at the actual code again in more detail

If you like SpringForward…

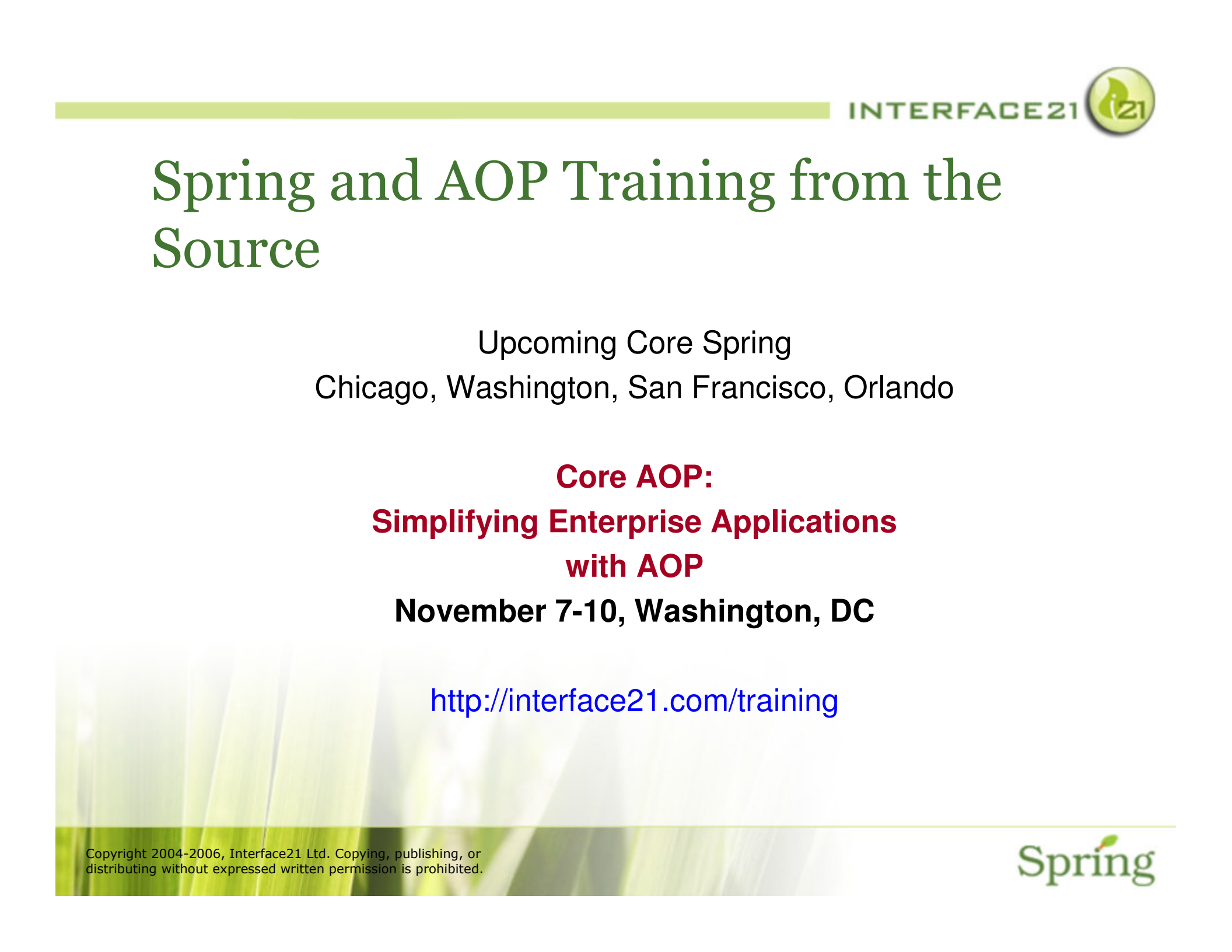
… you'll love The Spring Experience

The Spring Experience 2006
December 7th – 10th, Hollywood Florida

by Interface21 and NoFluffJustStuff Java Symposiums

- World-class technical conference for the Spring community
- Experience 3 full days, 55 sessions across 5 tracks
    1. Core Spring 2.0
    2. Core Enterprise 2.0
    3. Core Web 2.0
    4. Domain-Driven Design
    5. Just Plain Cool
- Enjoy five-star beach resort and amenities
- Converse with core Spring team and industry experts
    - Rod Johnson, Adrian Colyer, Ramnivas Laddad, Juergen Hoeller
    - Eric Evans, Luke Hohmann, Eamon McManus, Kathy Sierra
- **Register at http://www.thespringexperience.com**

THE SPRING EXPERIENCE
December 7-10, 2006
Hollywood, Florida



Spring and AOP Training from the Source

Upcoming Core Spring

Chicago, Washington, San Francisco, Orlando

Core AOP:

Simplifying Enterprise Applications with AOP

November 7-10, Washington, DC

http://interface21.com/training

Spring

- Questions?