Spring Web Flow
Dialogs for the Web

Keith Donald

Principal
Interface21
http://www.interface21.com

TS-3456

In the Next 60 Minutes…

You will learn how to orchestrate controlled web application conversations using Spring Web Flow.

Agenda

Problem

Approach

Usage examples

Integration

Future

Agenda

Problem

Approach

Usage examples

Integration

Future

Problem

Web applications are a mixed bag

- Consist of free navigations
  - Browsing a product catalog
  - Viewing product details
- And controlled page flows
  - Completing a checkout process
  - Applying for store credit

Free Navigation

Characteristics

- A set of pages connected by links
- Each link accesses a public resource
  - http://www.spring-shoes.com/catalog
  - http://www.spring-shoes.com/catalog/nb/476
- Users have access to each link freely
  - Links are often bookmarked
- **There is no controlled page flow**
- **There is no task to complete**

Controlled Page Flow

Characteristics

- A user task consisting of multiple steps
  - Has a starting point
  - Usually has an ending point
- Each task is accessible as a public resource
  - http://www.spring-shoes.com/checkout
- A task guides a single user toward completion of a business goal
- The progress of one user's task execution is independent of other users

DEMO

Real-world examples

Controller Characteristics

Free vs. controlled navigation

- A free navigation controller is simple
  - Stateless
  - Renders the view of a resource when requested
  - Existing frameworks do a good job here
- A controlled page flow controller is more complex
  - Stateful
  - Orchestrates a task with a linear progression
  - Renders views as necessary to allow the user to participate in the task
  - Not the focus of most existing frameworks

Controlled Navigation Challenges

What is traditionally difficult

- Enforcing a linear progression
  - Preventing the user from jumping around
  - Preventing the same task from being completed twice
- Managing state
  - Storing and accessing task state
  - Cleaning up the state of ended or expired tasks
  - Keeping server state in sync with the client
  - Preventing server state from being overwritten by other tasks executing in parallel

Agenda

Problem

Approach

Usage examples

Integration

Future

Enforcing a Linear Progression

Conventional approach

- The client drives the progression
  - Navigation hints are often embedded in URLs
    - `order.do?_currentPage=3`
    - `order.do?_finish=true`
- The controller validates that the client does the right thing according to the flow navigation rules
  - Figures out what step the client says she is at
  - Ensures task steps are executed in the correct order

DEMO

Enforcing a linear progression

Conventional

Enforcing a Linear Progression

Conventional implications

- The client can attempt to short-circuit the flow
  - Maliciously or accidentally
    - `order.do?_confirmed=true`
- The controller must prevent this
- As a result both the client and controller are often aware of flow navigation rules
- This often leads to:
  - Hard coded navigation hints in your JSPs
  - Many if/else statements within your controller implementation

Spring

WEBFLOW

Enforcing a Linear Progression

Spring Web Flow approach

- The controller drives the progression not the client
- The client simply provides the controller input when asked
  - **Client is not navigation rule aware**

```
client: start task
server: start; process input; render the starting form
client: submit
server: resume; process input; render the next form
client: submit
server: resume; finish; render confirmation
```

DEMO

Enforcing a linear progression

Spring Web Flow

Enforcing a Linear Progression

Spring Web Flow benefits

- The client can not short-circuit the flow
  - She can only provide the flow input from a specific point when asked
- The controller always knows what step the client is at
  - You no longer have to figure this out
  - You get a callback to resume processing from the correct point
- All flow navigation rules are encapsulated within the controller
  - Changing navigation rules does not impact clients

Managing State

Conventional approach

- The controller is stateless
- Stores task context in the session
- Cleans up context in the session after task completion
- Manages a session token to prevent completing the same task execution more than once

Code Sample

```
public Forward onFormSubmit(HttpServletRequest request) {
    if (isStartRequest(request)) {
        assertTaskNotInProgress(request);
        createTaskContext(request);
        return startingForm(request);
    } else if (isResumeRequest(request)) {
        assertSessionToken(request);
        if (isCurrentForm(request)) {
            updateSessionData(request);
            return errors(request) ? currentForm(request)
                   : nextForm(request);
        } else {
           return handleOutofSyncSubmit(request);
        }
    } else if (isFinishRequest(request)) {
        assertSessionToken(request);
        processSubmit(request);
        cleanupSessionData(request);
        removeSessionToken(request);
    }
}
```

Managing State

Conventional implications

- Use of the back button refers to session state captured at later point
- Opening a new window overwrites the other window's data
- Not properly cleaning up after task completion brings consequences
  - Memory leaks
  - Duplicate submission
  - Including stale data in a new task execution
  - Flow short circuit

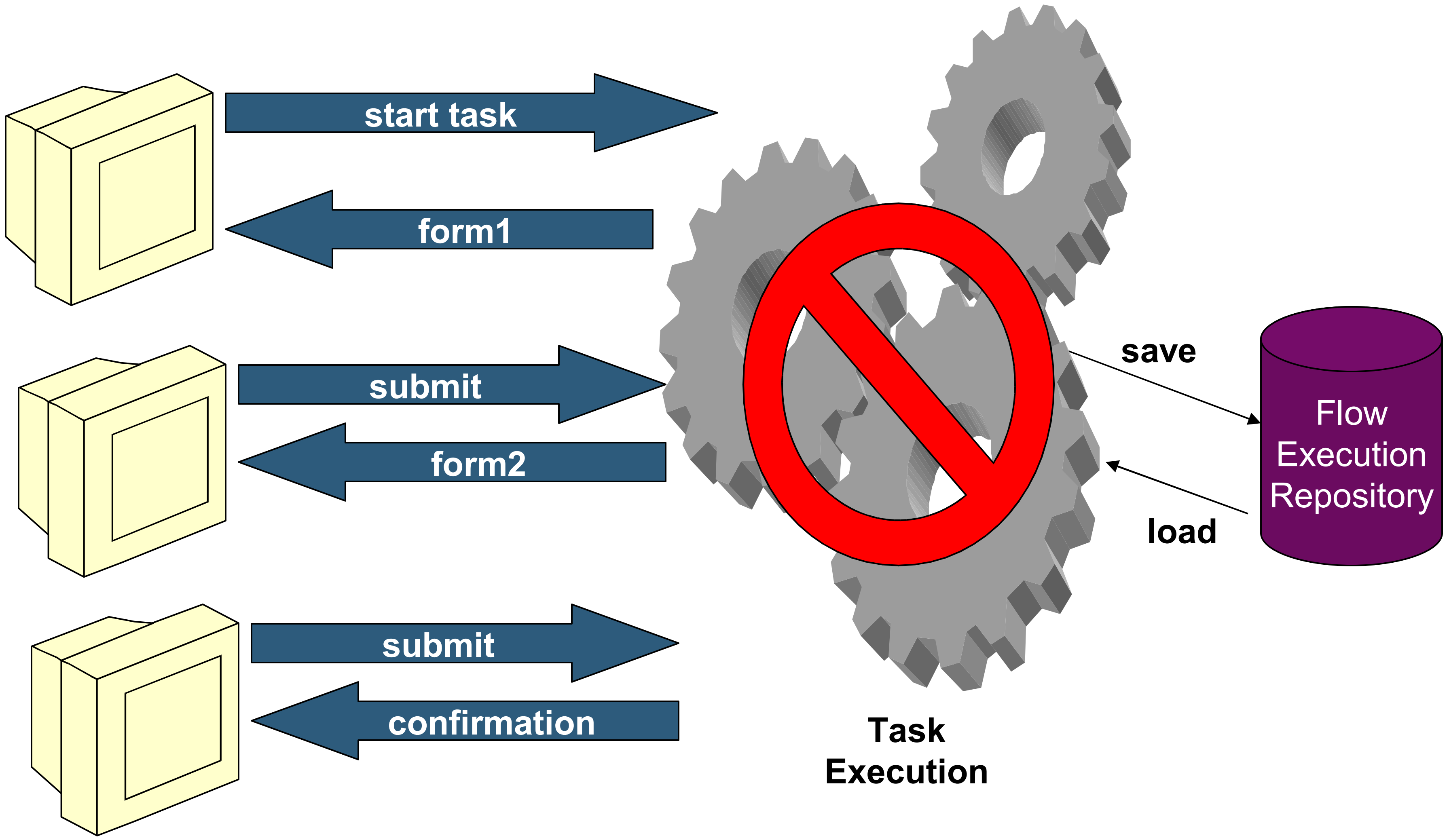
Spring

WEBFLOW

Managing State

Spring Web Flow approach

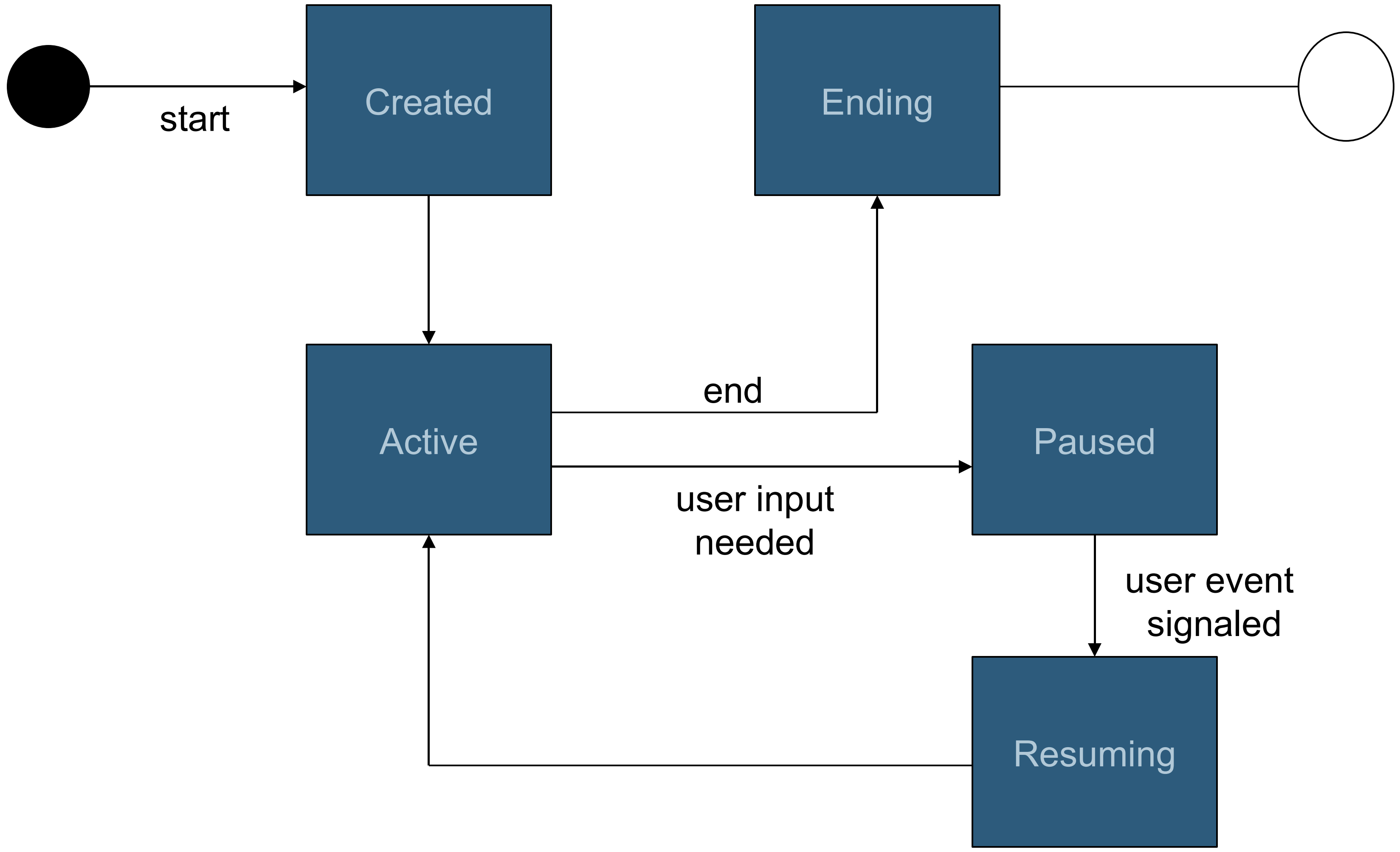
- The controller is stateful
  - Represents an executing task at a point in time
- Stored in a repository between requests
- Clients resume the controller to continue task execution from a point in time


start task
form1
submit
form2
submit
confirmation
save
load
Flow Execution Repository
Task Execution

Managing State

Spring Web Flow benefits

- Use of the back button refers to the state of the task execution at that point in history
- Opening a new window clones an independent task execution at the current step
- When a task completes it is purged from its repository
  - All managed state is eligible for garbage collection
  - It is impossible to continue a task that has completed


start
Created
Ending
Active
end
user input needed
Paused
user event signaled
Resuming

Approach Summary

Spring Web Flow vs. Conventional

- One controller, the flow, drives the entire task execution
- The flow pauses when client input is required
- The flow resumes when client input is provided
  - Initiated by an event
- Event processing logic is encapsulated within the flow
  - Client has no knowledge of flow navigation rules
    - Can only influence navigation via an event model, can not drive navigation

Agenda

Problem

Approach

Usage examples

Integration

Future

Flow Definition

How do you define a flow?

- You use a domain-specific language (DSL)
  - XML form is most popular

XML Representation

```
<flow start-state=“step1”>

    <my-state id=“step1”>
        <transition on=“event” to=“step2”/>
    </my-state>

    <my-state id=“step2”>
        <transition on=“event” to=“finish”/>
    </my-state>

    <end-state id=“finish”/>

</flow>
```

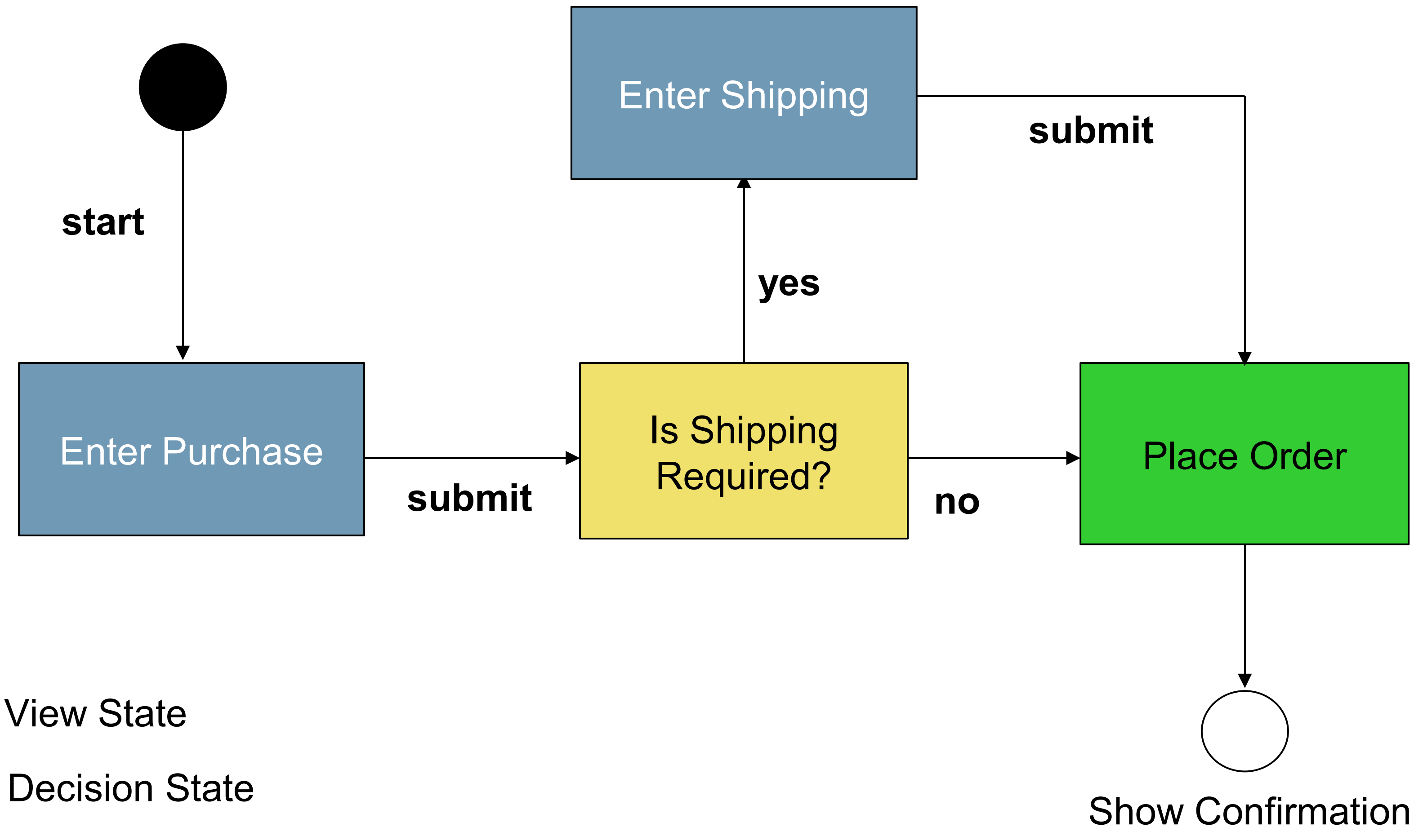
Flow Builder API

```
FlowBuilder builder = new AbstractFlowBuilder() {
    protected void buildStates() {
        addMyState(“step1”, on(“event”, to(“step2”));
        addMyState(“step2”, on(“event”, to(“finish”));
        addEndState(“finish”);
    }
}
FlowAssembler assembler =
    new FlowAssembler(“myFlow”, builder);
assembler.assembleFlow();
Flow flow = builder.getResult();
```

Flow Definition

Characteristics

- Declarative instructions to an execution engine
- A set of states that you define
- Each state executes a behavior when entered
  - View states solicit user input
  - Action states execute commands
  - Decision states make routing decisions
  - Subflow states spawn child flows
  - End states terminate flows
- Events you define drive state transitions
  - Transitions define the paths through the flow



🟦 View State

🟨 Decision State

🟩 Action State

○ End State

```
<flow start-state="enterPurchase">

    <view-state id="enterPurchase" view="purchaseForm">
        <transition on="submit" to="shippingRequired">
            <action bean="form" method="bindAndValidate"/>
        </transition>
    </view-state>

    <decision-state id="shippingRequired">
        <if test="${purchase.shipping}"
            then="enterShipping" else="placeOrder"/>
    </decision-state>

    <action-state id="placeOrder">
        <action bean="orderClerk"
                method="placeOrder(${purchase})"/>
        <transition on="success" to="showConfirmation"/>
    </action-state>

    <end-state id="showConfirmation" view="confirmation"/>

    <import resource="purchase-flow-beans.xml"/>

</flow>
```

Bean id to Implementation Binding

```
purchase-flow.xml
<action bean="orderClerk"
        method="placeOrder(${purchase})"/>


purchase-flow-beans.xml
<beans>
    <bean id="orderClerk" class="example.StubOrderClerk"/>
</beans>
```

- Spring Web Flow can bind to any method on any object:

```
public interface OrderClerk {
    OrderConfirmation placeOrder(Purchase purchase);
}
```

...Without your object depending on SWF APIs

Flow Definition

Benefits

- One artifact defines all task controller logic
- Is abstract; not concerned with:
  - State management
  - Servlet or Portlet APIs
  - URLs
  - Back button
  - Malicious clients
- The execution system cares for those concerns
- **A flow definition defines a task executable in any environment**

DEMO

The same flow executing within a Servlet and Portlet environment

Steps to Flow Execution

Readying a flow for execution

- Deploy your flow definitions to a registry:

```
<beans>

    <xmlFlowRegistry id="registry"
         flowLocations="/WEB-INF/flows/**/*.xml"/>

    …

</beans>
```

- By default a flow is assigned a registry identifier by convention
    - `purchase-flow.xml` becomes `purchase-flow`

Steps to Flow Execution

Readying a flow for execution

- Configure the flow executor for the environment you are running in
  - Spring MVC, JavaServer™ Faces, Struts supported out-of-the-box
- (Optional) Configure a strategy for how flow executions will be persisted between requests
  - In the session
  - To the client
- (Optional) Configure how flow executor arguments are extracted from the request
  - From request parameters
  - From the request path

Spring MVC Flow Executor

```
<beans>

    <flowController name="/*"
         registry-ref="registry"
         storage="client"
         argumentExtractor="requestPath"/>

</beans>
```

- Exposes flows in the registry for execution
- Uses request path parameterization to launch new flow executions
  - http://localhost/app/purchase
  - http://localhost/app/credit

Registry identifier

Flow Execution Rendering

Requirements

- View selections made by your flows must be resolvable to a response writer
- Typically a view template
    - Template resolution is handled by the framework SWF is integrating with
        - ViewResolver (Spring MVC)
            - Supports JavaServer Pages™ technology, Velocity, Freemarker, and custom views
        - Action forward (Struts)
        - View Name (JavaServer Faces technology)
- View templates must output the flow execution key to support a resume operation on submit

Example Template (JSP™ Technology)

```
<form method="post" action="${flowUrl}">
  …
  <spring-webflow:flowExecutionKey/>
  <input type="submit" name="_eventId_submit"
                       value="Submit">
</form>
```

- Flow execution key identifies a FlowExecution in the repository
  - Continues the conversation from the `view-state` that selected this view
- Event id communicates what user action occurred
  - Drives a transition out of the current `view-state`

Agenda

Problem

Approach

Usage examples

Integration

Future

Integrating Into Other Frameworks

Through an adaption layer

- Struts
  - FlowAction executes all flows
  - View selections are mapped to action forwards
  - An action form adapter allows SWF data binding
- JavaServer Faces platform
  - FlowPhaseListener restores flow executions from the repository on “restore view” phase
  - JSF components resolve flow expressions
    - Via FlowVariableResolver and FlowPropertyResolver
  - FlowNavigationHandler continues flows
- **Spring Web Flow is positioned as an embeddable page flow engine**

JavaServer Faces Integration Example

```
<faces-config>
    …
    <navigation-handler>
        o.s.webflow.executor.jsf.FlowNavigationHandler
    </navigation-handler>
    <property-resolver>
        o.s.webflow.executor.jsf.FlowPropertyResolver
    </property-resolver>
    <variable-resolver>
        o.s.webflow.executor.jsf.FlowVariableResolver
    </variable-resolver>
    …
    <phase-listener>
        o.s.webflow.executor.jsf.FlowPhaseListener
    </phase-listener>
    …
</flow>
```

JavaServer Faces Integration Example

- **Launching a flow as a command link**

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

- **Resuming a flow with component binding expressions**

```
<h:form id="form">
    ...
    <h:inputText id="propertyName"
       value="#{managedBeanName.propertyName}"/>
    ...
    <h:commandButton type="submit" action="submit"/>
</h:form>
```

Agenda

Problem

Approach

Usage examples

Integration

Future

Future

Spring Web Flow roadmap

- Nested, parallel flow executions
- JMX™-based flow execution management
  - Monitor in-flight conversations
- Conversation history subsystem
  - To support bread crumbs, statistics
- More integration
  - Tapestry
  - Business process management (BPM)
  - Acegi Security
  - Persistence providers (Session per flow)
  - Others?

Getting Started

Spring Web Flow jumpstart

- Access http://www.springframework.com/download
- Download Spring Web Flow 1.0 RC2
- Extract zip archive
- CD to `projects/build-spring-webflow`
- Execute `ant samples` to build sample apps
- Deploy sample .WARs for evaluation
  - Each sample is importable as a Eclipse project for easy review

Additional Resources

Spring Web Flow Related

- Reference and API documentation
  - http://www.springframework.org/documentation
- Support forum
  - http://forum.springframework.org
- Books
  - Expert Spring MVC and Web Flow, Apress
- Confluence Wiki
  - http://opensource2.atlassian.com/confluence/spring/display/WEBFLOW

Q&A

Spring Web Flow Dialogs for the Web

Keith Donald

Principal
Interface21
http://www.interface21.com

TS-3456