Building Enterprise Applications With Spring

Keith Donald
http://www.springframework.com
keith@interface21.com

Agenda
- The Spring architecture end-to-end
- Applied to a realistic business case
- You’ll learn the value Spring provides:
  - In the middle tier
  - In the web tier
- Goal: experience why enterprise Java development is fun again

Spring Is...
- The leading full-stack application framework applicable:
  - Everywhere—
    - System configuration and assembly
  - In the Middle tier—
    - Java/J2EE support libraries focused on ease-of-use
  - In the Web tier—
    - Model 2 MVC, Web Flow
  - On the Desktop—
    - A Swing-based rich client platform
- You use what you need
Many view Spring increasingly as an integration platform of best-of-breed components...

People like Spring because it...

- Is not invasive
- Brings leverage
- Lets you focus
- Provides choices in a consistent manner
- Delivers on power and convenience
- Is sustainable
  - Interface21, BEA, Oracle all provide professional Spring support

Spring Point of Sale (POS)™

- The best way to learn is by example
- Retail industry, order provisioning module
- This vertical slice must:
  - Allow a sales rep
  - To place a product order
  - On behalf of a customer
- Goal: implement this “end-2-end” slice in the next 45 minutes
How can Spring help in the middle tier?

- Managing your business objects
- Making your business objects testable
- Executing data access operations
- Demarcating transactions
- Performing security checks
- Messaging other systems (JCA, web, etc)

What are the key business contracts?

```java
@Transactional
public interface SalesProcessor {
    public void processSale(Sale sale);
}
```

```java
public class Sale {
    private int itemCount;
    private MonetaryAmount price;
    private ShippingType type;
    private CustomerKey customer;
    public MonetaryAmount discountRate();
    public MonetaryAmount totalCost();
}
```

Managing your business objects

- What are the implementations of those contracts?
  - public class JdbcSalesProcessor
- What does that implementation need “to work”?
  - A DataSource
  - Jdbc Helper
  - A Transaction Manager
Managing your business objects

Spring drives object assembly

```
<beans>
  <import resource="transaction-proxy-creators.xml"/>
  <bean id="salesProcessor" class="sellitem.JdbcSalesProcessor">
    <property name="dataSource" ref="dataSource"/>
  </bean>
  <bean id="dataSource" class="org.apache.dbcp.BasicDataSource">
    <property name="driverClassName" value="${db.driver}"/>
    <property name="url" value="${db.url}"/>
    <property name="username" value="${db.username}"/>
    <property name="password" value="${db.password}"/>
  </bean>
</beans>
```

Your business objects are J2SE POJOs focused on “main line” logic

- They are handed what they need “to work”
  - They don’t ask for it
  - So they don’t hard code dependencies on expensive resources
- What they need to work can be mocked to test main line logic in isolation

Making your business objects testable

- Your business objects are J2SE POJOs focused on “main line” logic
- They are handed what they need “to work”
  - They don’t ask for it
  - So they don’t hard code dependencies on expensive resources
- What they need to work can be mocked to test main line logic in isolation
Unit testing

- Unit tests test an object implementation in isolation
- No need for the Spring container
- Just extend TestCase, use the new operator to create objects to test
- Use mocks if necessary

Unit Testing - Example

```java
public class SaleUnitTests extends TestCase {
    public void testCalculateDiscountRate() {
        MonetaryAmount amt = new MonetaryAmount(100);
        Sale sale = new Sale(5, amt);
        sale.setCategory(Category.A);
        assertEquals(new MonetaryAmount(20), sale.getDiscountRate());
    }
}
```

Integration Testing

- Integration tests test object interaction to complete a use case
- Spring's container handles system test configuration
- Each test runs in its own transaction
- With automatic transaction rollback on test tear down
- Testing is fast
  - No need to deploy to a J2EE container
Integration Testing - Example

```java
public class SellItemIntegrationTests extends AbstractTransactionalDataSourceSpringContextTests {
    private SalesProcessor salesProcessor;

    @Override
    public String[] getConfigLocations() {
        return new String[] {
            "sellitem/middle-tier-configuration.xml"
        };
    }

    @Transactional
    public void testProcessSaleSuccess() {
        // main line test logic here
    }
}
```

Executing data access operations

- Resource connection management and statement preparation is handled for you
  - No more evil "TCFTC"
  - No more resource leaks
- Exception root cause analysis is handled for you
- All data access operations participate in transactions automatically

JDBC Data Access - Example

```java
public class JdbcSalesProcessor extends JdbcDaoSupport implements SalesProcessor {
    public void process(Sale sale) {
        getJdbcTemplate().executeUpdate("insert into SALES values (?,?,?,?,?)",
            null, sale.getItemCount(), sale.getPrice(), sale.getShippingType(),
            sale.getCustomerKey().longValue());
    }
}
```

Notice the code you are not writing...
Demarcating Transactions

- You express the TX policies you need, Spring makes it happen
  - TX management is an aspect
- The transactional context propagates to individual DAO operations seamlessly
- The TX manager is pluggable
- Scaling up from JDBC-driven to JTA-driven transactions is a matter of configuration
  - Does NOT impact application code
  - Does NOT impact demarcation metadata

```
<bean class="o.s.DefaultAdvisorAutoProxyCreator"/>
<bean class="o.s.TransactionAttributeSourceAdvisor">
  <property name="transactionInterceptor" ref="txInterceptor"/>
</bean>
<bean id="txInterceptor" class="o.s...TransactionInterceptor">
  <property name="transactionManager" ref="txManager"/>
  <property name="transactionAttributeSource">
    <bean class="o.s...AnnotationsTransactionAttributeSource"/>
  </property>
</bean>
<bean id="txManager" class="o.s...DataSourceTransactionManager">
  <property name="dataSource" ref="dataSource"/>
</bean>
```

How can Spring help in the web tier?

- Managing your web tier objects
- Making your web tier objects testable
- Modeling the page flow
- Managing conversational state
- Routing requests to handlers
- Binding form input to your domain objects
- Validating your domain objects, with error reporting
- Integrating multiple view technologies
Hosting your web-tier services

- You can “spring to life” your web tier, too
- Spring glues your web-tier with your middle-tier—seamlessly
- Examples of web-tier services:
  - Controllers
  - Flows
  - View Resolvers
  - Message Sources

Flow Architecture

- A wizard for a phone operator to use to sell items to her customers
- Characteristics:
  - An “application transaction” or “conversation” 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
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 lack a key abstraction: the Flow
- A Flow is typically longer than a single request but shorter than a session: a conversation!

Flow Conversation Analogy

Keith (on his European vacation)

Friend (Melbourne, Florida)

Keith:

Hi Friend!

Friend:

Hi Keith! So, have you decided to just stay in Europe for good now?

Keith:

Life IS great here, huh?

Friend:

Gotta go, bye!

The SWF approach

store

Flow Execution

Storage

FlowExecution

"Sell Item"
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

- Created
- Active
- Ending
- Resuming
- Paused

Start

- User input needed
- User event signaled
- Subflow spawned

End

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

The “Sell Item” Flow Definition
• If viewing on-line, see presentation notes

The Flow Integration Test
The Views (JSPs)
Demo of the Flow Execution
Advantages

- The logical flow is clearly defined, readable
- Navigation rules are encapsulated in one place
- No dependency on any request URLs
- 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
- HTTP independent
- Flow definitions are extremely toolable

Tying it all together

- User starts a new “sell item” process
- A new conversation is started, a new “Sale” object is put in “flow scope”
- User completes forms, the flow drives navigation and updates “Sale” state
- On completion of the flow, processSale is invoked
- processSale begins a transaction that propagates across multiple DAO calls
- End-2-end in very few lines of code

Resources and Upcoming Events

- www.springframework.org
  - New CMS portal means we now bring you a lot more content
- Web Flow Wiki provides a “Quick Start”
  - Practical guide
  - Articles
- Web Flow Ships with 7 sample applications
  - Phone Book (Core sample)
  - Sell Item (Wizard with Continuations)
  - Birth Date (Struts integration)
  - Item List (Transaction Synchronization)
  - Number Guess (Example of conversation history)
  - Flow Launcher (How to launch flows)
  - File Upload (A flow to upload files)
Resources and Upcoming Events

- **Public Spring Training**
  - 4-day “bootcamps” across North America, Europe
  - [www.springframework.com/training](http://www.springframework.com/training)

- **The Spring Experience**
  - International conference for agile Java developers
  - [http://www.thespringexperience.com](http://www.thespringexperience.com)
  - December 7 – 10th, 2005, Sheraton Resort, Bal Harbour Beach Florida
  - 40+ technical sessions across four tracks: Web, Enterprise, Process, Beyond J2EE
  - Featuring
    - Rod Johnson (Spring Founder, i21 CEO)
    - Juergen Hoeller (Spring Co-Founder, i21 CTO)
    - Adrian Colyer (AspectJ Founder, i21 Chief Scientist)
    - Rob Harrop (Author, Pro Spring)
    - ... and many more
  - Super early bird registration period open now

---

Conclusion

- Spring is a complete, modular, pragmatic full-stack application framework
- It adds serious value in:
  - The middle tier
  - The web tier
- Spring will continue to innovate
- Spring will continue to simplify
- **Got Spring?**