Strecks Java 5 Extensions

Presented by Phil Zoio, Realsolve Solutions

March 17, 2006

Agenda

- Background and introduction
- The Details
  - Form validation
  - Data binding and conversion
  - Actions
  - Interceptors
  - Navigation
- Concluding comments

About Struts

- Most successful Java web framework
- First widely adopted MVC Model 2 framework
- Proven on thousands of projects

Ground-breaking in its time but these days:

- Not universally popular
- Overtaken by other frameworks

Various Struts initiatives

  - Shale
  - Struts Action 2 (WebWork integration)

Some Background

- In-depth experience last year with
  - Component-based frameworks (Tapestry/JSF)
  - Dependency injection (Spring)
  - Java 5
- Requirement to build new web application in Struts
  - Initially reluctant – didn't want to forgo advanced features
  - Decided to add features to existing Struts
- After completion of project
  - Internal refactoring
  - Additional features added
  - To be open sourced
  - Provisional name Strecks

About Strecks Java 5 Extensions

- Built on existing Struts 1.2.x code base
- Main features include
  - Pure POJO action beans
  - Action dependency injection
  - Action controllers
  - Interceptors
  - Form validation using annotations
  - Data conversion and binding using annotations
  - Pluggable navigation

Strecks design goals

- offer or even improve on features offered by competitive frameworks
- simplify, not replace, existing Struts programming model
  - no major high level architectural changes
  - actions, action forms, etc. still present, but enhanced
  - UI and configuration unchanged
- don't air your dirty linen in public
  - users don't want to see framework internals
- introduce no compatibility issues apart from Java 5
- keep easy to learn for existing Struts users

Strecks - The Details

Form Validation

Form Validation in Struts 1.2

- Manual validation code verbose and tedious
- XML-based validations using *Validator* framework not ideal:
  - Large validation file
  - Verbose format “XML Hell”

Form Validation – Struts 1.2 Example

```
if (days == null)
{
    ActionMessage error =
      new ActionMessage("holidaybookingform.days.null");
    errors.add("days", error);
    hasError = true;
}
else
{
    if (!GenericValidator.isInt(days))
    {
        ActionMessage error =
              new ActionMessage("holidaybookingform.days.number”);
        errors.add("days", error);
        hasError = true;
    }
}
```

Form Validation in Strecks

- Still uses `ActionForm`
- Interface between view layer and action forms unchanged
- Validation (and binding) layer within `ActionForm`
  - Implement marker interface `AnnotatedForm`
  - Add `@Validate...` annotations to setters
  - `validate()` method will still be called for additional (manual validation)
- Application not exposed to framework internals

Form Validation – Strecks Example

```
@ValidateRequired(key = "holidaybookingform.days.null")
@ValidateInteger(key = "holidaybookingform.days.number")
public void setDays(String days)
{
    this.days = days;
}
```

Adding Form Validators

Easy to add validators

- Validator implements `Validator` interface

```
public boolean validate(T value);
```

- `Validator` instance created by `ValidatorFactory`
- `ValidatorFactory` class identified by annotation
- Validation parameters passed in via annotation

Called the "*Annotation Factory*" pattern

Consequence: full extensibility with

- No XML
- No existing file changes

Data Binding And Conversion

Binding and Conversion in Struts 1.2

- Use of richly typed `ActionForms` tricky
- No per-field control of type conversion
- Manual data binding code verbose and tedious

Bind and Conversion – Struts 1.2 Example

```
public void readFrom(HolidayBooking booking)
{
    if (booking != null)
    {
        if (booking.getStartDate() != null){
            this.startDate = new
   java.sql.Date(booking.getStartDate().getTime()).toString();
        }
    }
}

public void writeTo(HolidayBooking booking)
{
    if (this.startDate != null && this.startDate.trim().length() > 0)
        booking.setStartDate(java.sql.Date.valueOf(startDate));
}
```

Binding and Conversion in Strecks

As with validation

- Binding and conversion layer within `ActionForm`
  - Implement marker interface `AnnotatedForm`
  - Add `@Bind...` and `@Bind...` to annotations getters
  - Add getters and setters for domain model objects
  - Controller will call bind handlers
- Application not exposed to framework internals

Bind and Conversion – Strecks Example

```
private HolidayBooking booking;

@BindSimple(expression = "booking.startDate")
@ConvertDate(pattern = "yyyy-MM-dd")
public String getStartDate()
{
    return startDate;
}
```

Adding Bind Handlers

- Bind handler implements `BindHandler` interface
- `BindHandler` instance created by `BindHandlerFactory`

```
public void bindInwards(Object src, String property);
public void bindOutwards(Object src, String property);
```

- `BindHandlerFactory` class identified by annotation
- Binding parameters passed in via annotation
- `Converter` class can be specified in annotation

Adding Stand-alone Type Converters

Converter can also be specified in own annotation

- Converter annotation in same method as bind annotation
- Converter implements `Converter` interface

```
public T toTargetType(S toConvert);
public S toSourceType(T toConvert);
```

- `Converter` instance created by `Converteractory`
- `ConverterFactory` class identified by annotation
- Converter parameters passed in via annotation

Decouples bind handler and converter definition

Used when converter must be parameterized

Dependency Injection

Dependency Resolution in Struts 1.2

- State held in request, session and context attributes
  - Verbose API
  - Type casting necessary
  - Not very object oriented
- Manual conversion of request parameters
- Service layer references obtained via programmatic hooks

Dependency Resolution – Struts 1.2 Example

```
public ActionForward execute(ActionMapping mapping, ActionForm form,
   HttpServletRequest request,
        HttpServletResponse response) throws Exception
{
   //use Spring ActionSupport superclass to get context
   HolidayBookingService service = (HolidayBookingService)
   getWebApplicationContext().getBean(
            "holidayBookingService");

    long id =
   Long.parseLong(request.getParameter("holidayBookingId"));
    HolidayBooking holidayBookings = service.getHolidayBooking(id);
    request.setAttribute("holidayBooking", holidayBookings);

    return mapping.findForward("success");
}
```

Strecks Dependency Injection

- Actions instantiated on per-request basis
- Dependencies resolved *declaratively* via annotations
- Support for:
  - *Typed* request parameters (using `Converter`)
  - Request, session, application context attributes
  - Spring beans, message resources, locale, `ActionForm`
  - etc...

Dependency Injection – Strecks Example

```
public String execute()
{
    HolidayBooking holidayBookings
        = service.getHolidayBooking(holidayBookingId);
    webHelper.setRequestAttribute("holidayBooking", holidayBookings);
    return "success";
}

@InjectSpringBean(name = "holidayBookingService")
public void setService(HolidayBookingService service) {
    this.service = service;
}

@InjectRequestParameter(required = true)
public void setHolidayBookingId(long holidayBookingId){
    this.holidayBookingId = holidayBookingId;
}
```

Adding Dependency Injection Mechanisms

- Injection handler implements `InjectionHandler` interface

```
public Object getValue(ActionContext context);
```

- `InjectionHandler` instance created by `InjectionHandlerFactory`
- `InjectionHandlerFactory` class identified by annotation
- Injection parameters passed in via annotation

Injection possible for any data obtained via request, response, application context, action mapping, action form

Actions and Controllers

Actions in Struts 1.2

- Actions must be thread-safe
  - No request-specific instance fields
  - Request dependency injection impossible
- Single inheritance hierarchy
- Difficult to reuse request processing logic

Actions in Strecks

Implemented by combination of action controller and action bean

- Controller implements common request workflow
- Action bean handles domain-specific processing tasks

Action Beans

- Pure POJOs
- Registered in struts-config.xml
- Identify controller using annotation
- Implement controller-defined interface
- Dependencies resolved via dependency injection

Any request processing pattern can be abstracted into a controller. Action beans handle the fine grained domain specific action interactions

Action Bean Example Outline

```
@Controller(name = BasicSubmitController.class)
public class SubmitEditBookingAction implements BasicSubmitAction
{

    public String cancel()
    {
        //implementation ommitted;
    }

    public String execute()
    {
        //implementation omitted
    }

    //injected properties

}
```

Action Beans Example Implementation

```
public String cancel()
{
  webHelper.setRequestAttribute("displayMessage", "Cancelled operation");
  webHelper.removeSessionAttribute("holidayBookingForm");
  return "success";
}

public String execute(){
  HolidayBooking holidayBooking = form.getBooking();
  holidayBookingService.updateHolidayBooking(holidayBooking);

  webHelper.setRequestAttribute("displayMessage", "Successfully updated
   entry: " + holidayBooking.getTitle());
  webHelper.removeSessionAttribute("holidayBookingForm");

  return "success";
}
```

Action Controllers

- Defines action bean's interface
- Implement request processing workflow
- Instantiates action bean per request
- Interacts with action bean through interface
- Single instance, holds no request state
- Various out the box implementations, including
  - form handling controllers
  - dispatch controllers
- Single controllers *reusable* across many actions

Controller Implementation Outline

```
@ActionInterface(name = BasicSubmitAction.class)
public class BasicSubmitController extends BaseBasicController
{
    @Override
    protected ViewAdapter executeAction(Object actionBean,
          ActionContext context) {

        BasicSubmitAction action = (BasicSubmitAction) actionBean;
        //omitted ... figure out whether form is cancelled
        if (form instanceof BindingForm && !cancelled)
        {
            //omitted ... do binding and call preBind();
        }

        String result = cancelled ? action.cancel() : action.execute();
        return getActionForward(context, result);
    }
}
```

Action Bean Annotations

- Framework for adding behaviour through annotations
  - `ActionBeanAnnotationReader` interface
- Extends contract between controller and action bean
- Fully extensible: no XML or source file changes
- Uses:
  - Reading dependency injections
  - Pluggable navigation
  - Action bean “source” configuration
    - `@SpringBean`
    - Singleton action beans *(planned)*
  - Dispatch method lookup
  - Action-specific interceptors *(planned)*

Interceptors

About Interceptors

Most web applications have common operations:

- Logging
- Authentication
- Custom state management
- etc.

In Struts 1.2, implemented either via:

- `RequestProcessor` subclasses
- Common `Action` base classes

Interference with inheritance hierarchy

Interceptors in Strecks

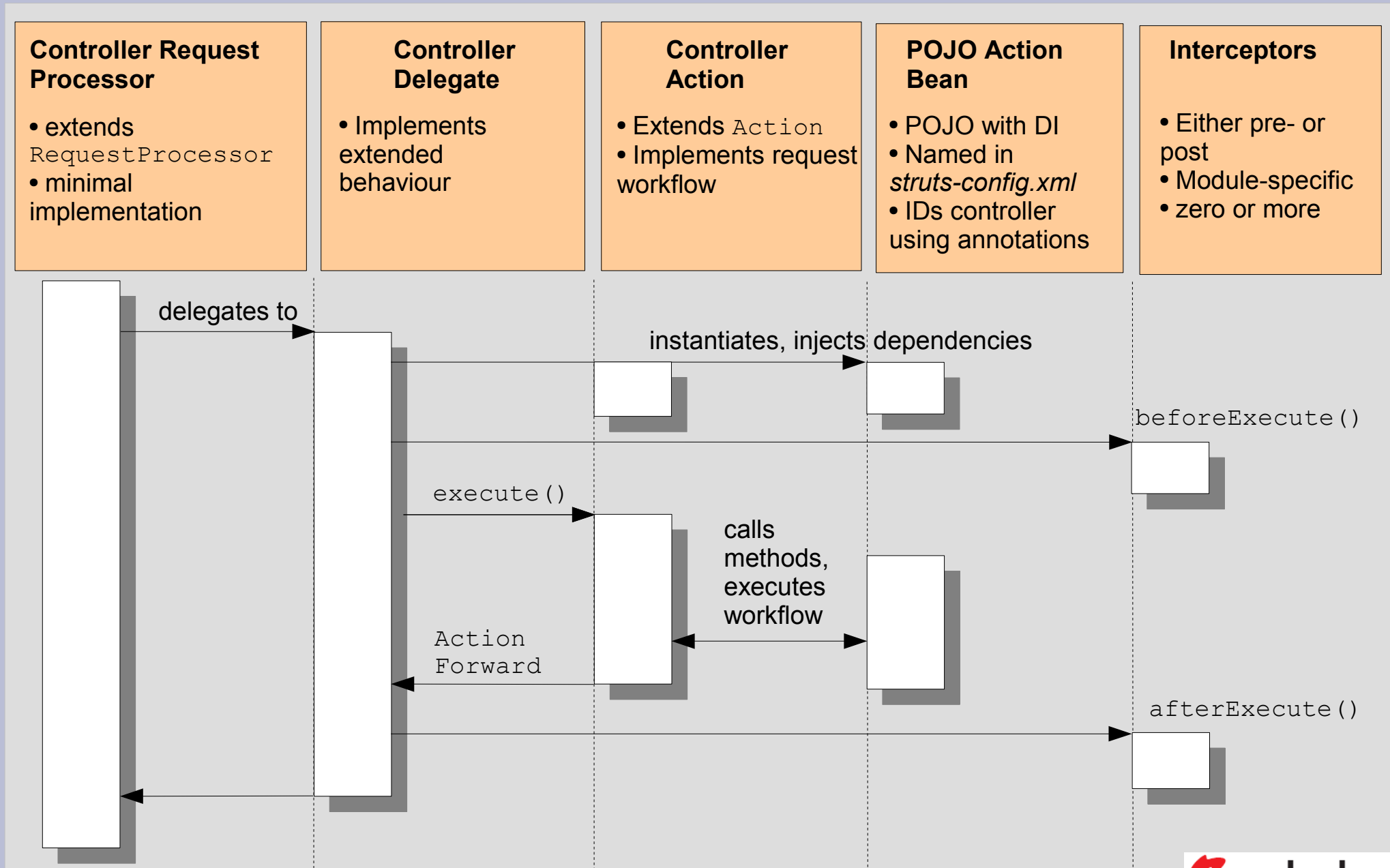
- Interceptor interfaces
  - `BeforeInterceptor`
  - `AfterInterceptor`
- Interceptors registered via configuration
- `BeforeInterceptor`
  - Called after dependency injection
  - Interrupt execution by throwing exception
- `AfterInterceptor` called before view handling
  - Exceptions simply logged
- Neither can return navigation result

Interceptor Example - Strecks

```
public class ActionLoggingInterceptor implements BeforeInterceptor,
   AfterInterceptor
{
    public void beforeExecute(Object actionBean, ActionContext context)
    {
        HttpServletRequest request = context.getRequest();
        log.info("Starting action for " + request.getRequestURI());
        log.info("Using " + actionBean.getClass().getName());
    }

    public void afterExecute(Object actionBean, ActionContext context)
    {
        HttpServletRequest request = context.getRequest();
        log.info("Ended action for " + request.getRequestURI());
    }
}
```

Struts Extensions Action Invocation



realsolve

Navigation

Navigation in Struts 1.2

- Navigation via returning `ActionForward`
- Can be created manually or via `actionMapping.findForward()`

But limited support for alternative view rendering

- Typically implemented:
  - Within action, with null `ActionForward` returned
  - Forwarding to external servlet (e.g *VelocityStruts*)

Navigation in Strecks

- Two sets of controller implementations
  - `public String method()` result gets `ActionForward` (outcome-based navigation)
  - other uses pluggable navigation via `@NavigateForward` annotation

  Convenient solution vs flexible solution

Basic Controller Navigation

```
@Controller(name = BasicController.class)
public class ExampleBasicAction implements BasicAction
{
    private String message;

    public String execute()
    {
        message = "Executed " + ExampleBasicAction.class.getName();
        return "success";   //
    }

    public String getMessage()
    {
        return message;
    }
}
```

Navigable Controller Navigation

```
@Controller(name = NavigableController.class)
public class ExampleNavigableAction implements NavigableAction
{
    private String message;

    public void execute()
    {
        message = "Successfully executed";
    }

    @Navigate
    public String getResult()
    {
        return "success";
    }

    public String getMessage(){ return message;    }
}
```

More on Navigation

- `Controllers` return `ViewAdapter` (not `ActionForward`)
  - `ActionForwardViewAdapter` class
  - `RenderingViewAdapter` interface
- `ViewAdapter` allows support for alternative view rendering mechanisms
  - Allows, for example, rendering via *Spring* views
  - AJAX, Remoting, XSLT, etc. easily supported
- Support for “page” classes for supporting formatting logic
- Support for redirect after post

Summary

Conclusion

Strecks's aim has been to:

- Match the simplicity of Struts
- Match the validation and type conversion power of JSF/Tapestry
- Match the flexibility of Spring MVC and WebWork

Who Should Use Strecks?

Good solution for enterprises which:

- Have invested in Struts
  - Developer knowledge
  - Existing applications
- Want to take advantage of powerful Java 5 features
- Want a framework supporting modern best practices
  - Ease of testability
  - Use of interfaces, design patterns & good OOP
  - dependency injection
- Want these features **now**
- BUT Don't want the pain of a bigger migration

Current Status

- Ready for open source release
- Nearly version 1.0 ready
- Should be announcement very shortly
- Any help would be appreciated ...