Direct Struts Actions 2 1.0 Component Specification

1. Design

The design implements the actions described in the Requirement Specifications. It provides a class for every action. The actions are usual Struts 2 actions: they contains the setters and getters for the http parameters and the execute method where the business logic is performed. The facades are injected externally: they are used to persist or retrieve the TopCoder entities. An interceptor is provided to re-use the annotation validation of Struts 2 and the validation classes of Struts Framework TopCoder component. A converter is also provided to permit to use all the data of the TopCoder entities.

1.1 Design Patterns

The patterns used in this design are:

- **Template method**: the BaseDirectStrutsAction delegate the business logic to the abstract executeAction method implemented by the sub classes; in the PayContestAction the logic is delegated to the abstract methods

1.2 Industry Standards

None

1.3 Required Algorithms

1.4 Validation

The validation is performed by the usual annotations of Struts 2 and custom validation. The ValidationErrorsInterceptor will put the validation messages of Struts 2 into the Validation errors.

The Struts 2 annotations validation requires a default message and an optional key. Use an explicit default message that contains the meaning of the error. For example if the field is required and it must be a string with length max of 5 characters and it's set to 10 characters the create the message as “The *field*'s length must be not empty and max 10 characters” where *field i*s the name of field. The messages are up to developer. Set also the optional key so the message can be changed without re-compile the project. The key must be constructed as: 'i18n.nameOfTheAction.nameOfFieldMeaningOfValidator”. For example in the previous case create the key 'i18n.saveDraftContestAction.contestNameRange'.

1.5 Component Class Overview

BaseDirectStrutsAction

This is the base action of all actions in this component. It sets the AggregateDataModel to the model using the prepare logic of struts framework (in the assembly/development the logic of this class could be implemented directly in AbstractAction and delegate the logic of the action to the template method.

It also manages the exceptions thrown by the template methods

ContestAction

This is a base class only to hold the contest service facade shared among several actions

PayContestAction

This is the base class for the payment actions. It holds the main logic.

StudioOrSoftwareContestAction

This is the base class for the action which must decide if a studio or a software contest is managed based on the project or contest id.

It contains the common logic.

PayByBillingAccountAction

This action will make a payment for user using an existing billing account, as per ARS 2.11.1.2-3. This will use the processContestPurchaseOrderPayment method of ContestServiceFacade. It will also send an email to the user (unfortunately the PM at this time didn't answer how to retrieve the mail, so this will be fixed in the final fixes)

PayByCreditCardAction

This action will make a payment for user using a credit card, as per ARS 2.11.1.2-3. This will use the processContestCreditCardPayment method of ContestServiceFacade.

It will also send an email to the user
GetDocumentsContestAction

This action will return all documents of the contest by getting the contest itself. The software documents are available in the SoftwareCompetition.AssetDTO, in the documentation field.

Studio documents are in UploadedDocument instances, and Software documents are in the CompDocument instances.
GetDocumentFileContestAction

This action permits to download a document file from the data provided in the request. It's based on the ARS 2.8.1.8 . The class will be annotated with the annotation Result(name = ActionSupport.SUCCESS, type = "stream") to permit the Struts 2 framework to return a stream result

FileUploadAttachContestFileAction

This action permits to upload and attach a document to a contest.

It uses the FileUploadInterceptor: it will fill the fields contestFile, contestFileContentType and contestFileName. It's related to ARS 2.8.1.3.
It's will persist the related UploadDocument
GetContestTechnologiesAction

This action will return all available technologies using ContestServiceFacade's getActiveTechnologies.

DeleteDocumentContestAction

This action will delete the document. It will first remove the document from the contest then delete the document itself. This is done using the ContestServiceFacade's removeDocumentFromContest then removeDocument methods, respectively

GetContestCategoriesAction

This action will get a contest with the given id. Use ContestServiceFacadeÔs getContest or getSoftwareContestByProjectId based on whether this is a studio or non-studio contest.

MimeTypeRetriever

This one is an useful class to retrieve the mime type from the file name and the mime type id. It uses the MimetypesFileTypeMap to retrieve the mime type. The ids are hardcoded.

1.6 Component Exception Definitions

None

1.7 Thread Safety

The custom validator and the custom converter classes are thread safe because they are stateless and the parent classes are thread safe. The actions are not thread safe but the thread safety is not necessary because in Struts 2 the actions are created in every request: so they can be mutable, they are performed by a single thread. So at the conclusion the component, respect to the environment, is thread safe.

2. Environment Requirements

2.1 Environment

- Java 1.5/J2EE 1.5
- JBoss 4.0.2
- Informix 11
- MySQL 5.1
- Struts 2.1.8.1
- Spring 3.0
- Javascript 1.8
- Mozilla Firefox 2.0/3.0
- IE 6.0/7.0
- Google Chrome
- Safari 3/4

2.2 TopCoder Software Components

- **Contest Service Façade 1.0**: used to manage contests, to retrieve the contests (studio or software) and related entities
- **Struts Framework 1.0**: used as base framework, the bases classes extend from it

NOTE: The default location for TopCoder Software component jars is../lib/tcs/COMPONENT_NAME/COMPONENT_VERSION relative to the component installation. Setting the tcs_libdir property in topcoder_global.properties will overwrite this default location.

2.3 Third Party Components

- **Struts 2.1.8** : http://struts.apache.org/2.x/

3. Installation and Configuration

3.1 Package Name

com.topcoder.service.actions

3.2 Configuration Parameters

The facades and other properties of the action must be injected externally. An unique name is not present, for these configuration parameters, they can be injected using annotations (in the assembly) or Spring configuration files.

The i18n validation message keys:

i18n key	validation	default message
i18n.DeleteDocumentContestAction.documentIdRequired	document id is required	The documentId can not be null
i18n.DeleteDocumentContestAction.documentIdRange	document id must > 0	The documentId must be greater than 0
i18n.DeleteDocumentContestAction.contestIdRequired	contest id is required	The contestId can not be null
i18n.DeleteDocumentContestAction.conestIdRange	contest id must > 0	The contestId must be greater than 0
i18n.FileUploadAttachContestFileAction.contestIdRequired	contest id is required	The contestId can not be null
i18n.FileUploadAttachContestFileAction.contestIdRange	contest id must > 0	The contestId must be greater than 0
i18n.FileUploadAttachContestFileAction.contestFileDescriptionRequired	file description is required	The contestFileDescription can not be null
i18n.FileUploadAttachContestFileAction.contestFileDescriptionLengthRange	file description's length must between 1 and 4096	The contestFileDescription's length must between 1 and 4096
i18n.FileUploadAttachContestFileAction.documentTypeIdRequired	documentTypeId is required	The documentTypeId can not be null
i18n.FileUploadAttachContestFileAction.documentTypeIdRange	documentTypeId must between 0 and 24	The documentTypeId must be greater or euqal than 0 and less than 25
i18n.GetDocumentFileContestAction.documentIdRequired	document id is required	The documentId can not be null
i18n.GetDocumentFileContestAction.documentIdRange	document id must > 0	The documentId must be greater than 0
i18n.PayByBillingAccountAction.poNumberRequired	poNumber is required	The purchase order number can not be null
i18n.PayByBillingAccountAction.poNumberLengthRange	poNumber's length must be at least 1	The purchase order number's length must at least 1 chars
i18n.PayByBillingAccountAction.clientIdRequired	clientId is required	The clientId can not be null
i18n.PayByBillingAccountAction.clientIdRange	clientId must > 0	The clientId must be greater than 0
i18n.PayByBillingAccountAction.projectIdRequired	projected is required	The projectId can not be null
i18n.PayByBillingAccountAction.projectIdRang	projected must > 0	The projectId must be greater than 0
i18n.PayByBillingAccountAction.typeRequired	payment type is required	The payment type can not be null

3.3 Dependencies Configuration

The TopCoder components and Struts 2 framework must be configured properly.

4. Usage Notes

None

4.1 Required steps to test the component

Extract the component distribution.

Follow the dependencies configuration.

Execute ‘ant test’ within the directory that the distribution was extracted to.

4.2 Required steps to use the component

See the demo section

4.3 Demo

At this point I set up the struts.xml to configure the action and the interceptor:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
    "http://struts.apache.org/dtds/struts-2.0.dtd">

<struts>
<constant name="struts.objectFactory" value="spring" />
<package name="demo" namespace="/" extends="struts-default">
        <action name="getContestCategories"
class="com.topcoder.service.actions.GetContestCategoriesAction">
                <result name="success" type="freemarker">/templates/categories.ftl</result>
        </action>
        <action name="getContestTechnologies"
class="com.topcoder.service.actions.GetContestTechnologiesAction">
                <result name="success" type="freemarker">/templates/technologies.ftl</result>
        </action>
        <action name="getContestDocuments"
class="com.topcoder.service.actions.GetDocumentsContestAction">
                <result name="input">/getDocuments_by_contestId.jsp</result>
                <result name="success" type="freemarker">/templates/documents.ftl</result>
        </action>
        <action name="getProjectDocuments"
class="com.topcoder.service.actions.GetDocumentsContestAction">
                <result name="input">/getDocuments_by_projectId.jsp</result>
                <result name="success" type="freemarker">/templates/documents.ftl</result>
        </action>
        <action name="fileUploadContestFile"
class="com.topcoder.service.actions.FileUploadAttachContestFileAction">
                <result name="input">/fileUploadAttachContestFile.jsp</result>
                <result name="success" type="freemarker">/templates/upload.ftl</result>
        </action>
        <action name="getDocumentsFile"
class="com.topcoder.service.actions.GetDocumentFileContestAction">
                <result name="input">/getDocumentFileContest.jsp</result>
                <result name="success" type="stream"></result>
        </action>
        <action name="payByCreditCard"
class="com.topcoder.service.actions.PayByCreditCardAction">
                <result name="input">/payByCreditCard.jsp</result>
                <result name="success" type="freemarker">/templates/result.ftl</result>
        </action>
        <action name="payByBillingAccount"
class="com.topcoder.service.actions.PayByBillingAccountAction">
                <result name="input">/payByBillingAccount.jsp</result>
                <result name="success" type="freemarker">/templates/result.ftl</result>
        </action>

</package>
</struts>
```

At the end I can create the related jsp page for deleting the document from contest:

```
<%@ taglib prefix="s" uri="/struts-tags" %>
<html>
    <head>
        <title>File Upload To Contest</title>
        <link href="<s:url value="/css/main.css"/>" rel="stylesheet" type="text/css"/>
    </head>
    <body>
        <h2>File Upload To Contest</h2>
        <div style="color:blue;">(contestId 10 is in mock data)</div>

        <s:actionerror />

        <s:form action="fileUploadContestFile" method="post" enctype="multipart/form-data"
validate="true">
                <s:textfield name="contestId" label="Contest Id"/>
                <s:file name="contestFile" label="File"/>
                <s:textfield name="documentTypeId" label="documentTypeId"/>
                <s:textfield name="contestFileDescription" label="contestFileDescription" />
                <s:submit value="Submit" align="center"/>
        </s:form>
    </body>
</html>
```

5. Future Enhancements

It's possible to implement other actions.