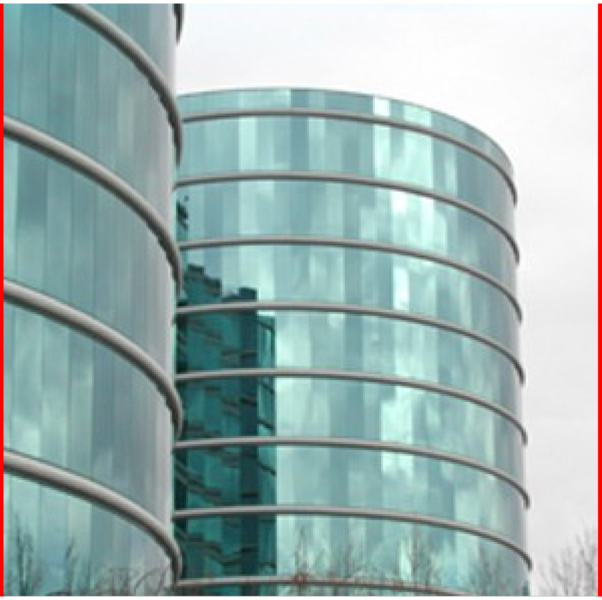


Hudson Web Architecture

Winston Prakash

The View Architecture

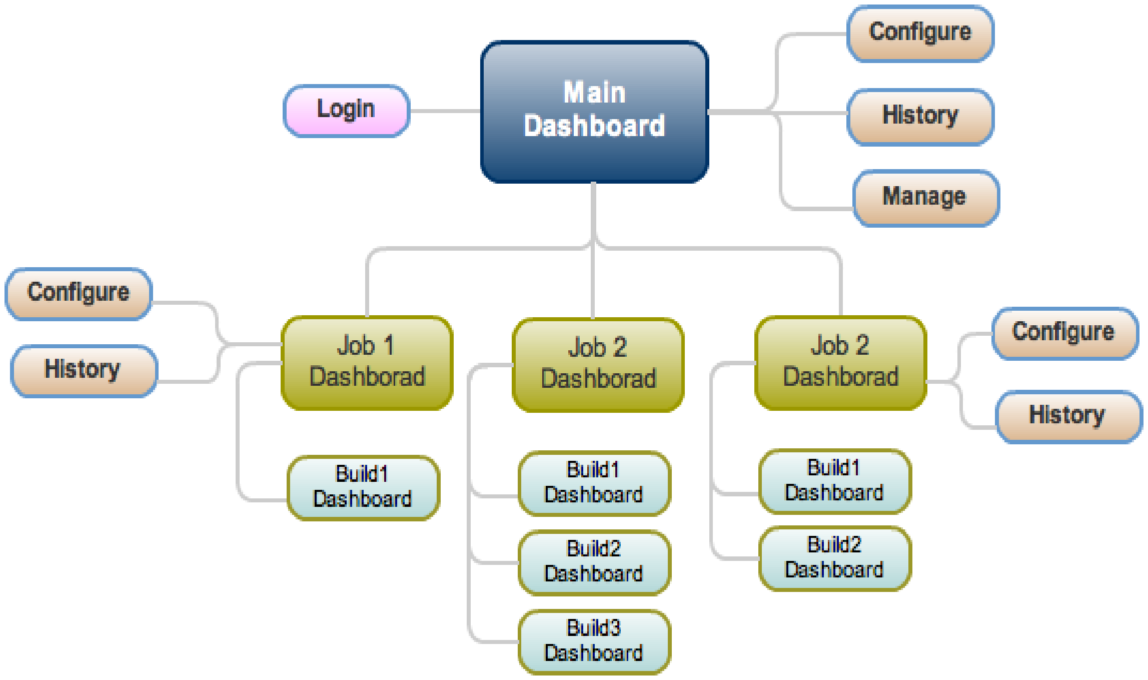
Hudson Web Client is the user facing View of the Hudson Application. Each Page or partial page of the Web Client corresponds to a Model Object and stitched together using stapler – an in-house REST Framework.

Hudson Client View has mainly three Dashboards

- Main Dashboard
- Project or Job Dashboard
- Build Dashboard

Each Dashboard displays

- Action links
  - Configure
  - Create
  - etc
- Dashboard specific sub views
  - Build History
  - Trends
  - etc

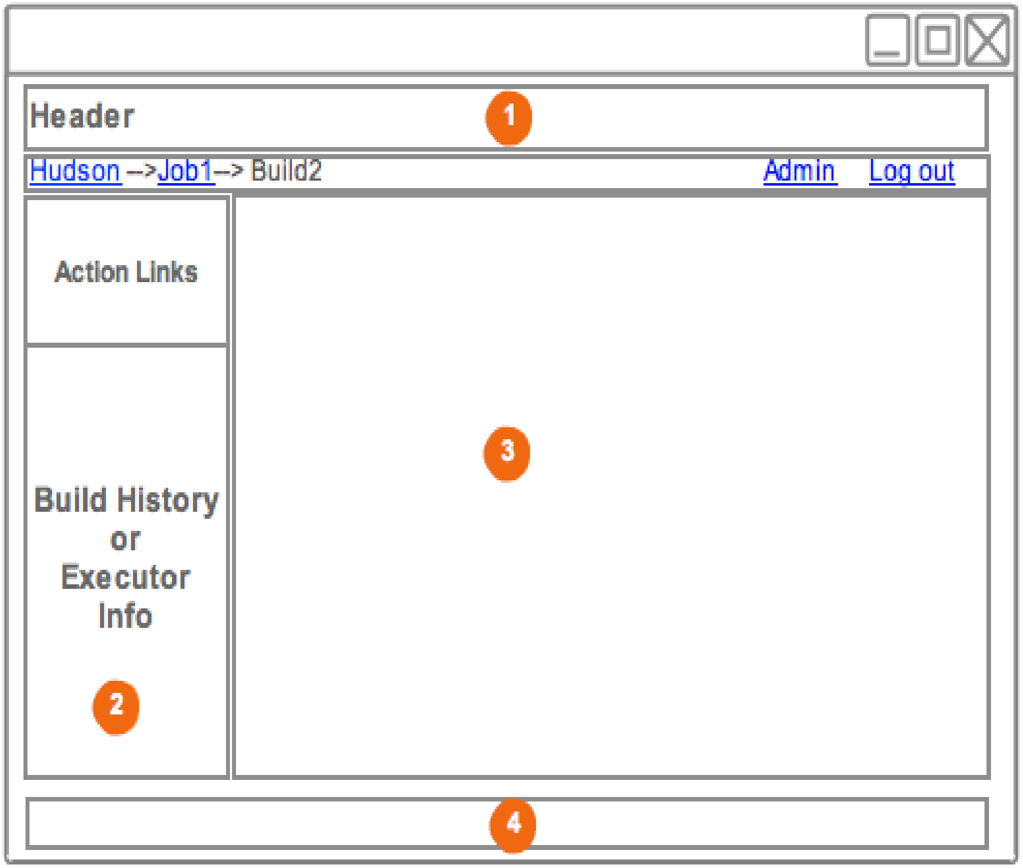


The Web Page

Each Hudson page is rendered from a Model object and associated Page Definition. The page definitions are stored in Jelly files and written using jelly tags.

The page is constructed from

- Header - Displays Breadcrumb and user related action links.
- Side Bar with two sections
  - Action links corresponding to that Dashboard
  - Ajax based sub view to display Build History or Executor status
- Content Pane where the dashboard specific contents are displayed
- Footer where the copyright related material are displayed.



Model View Relationship

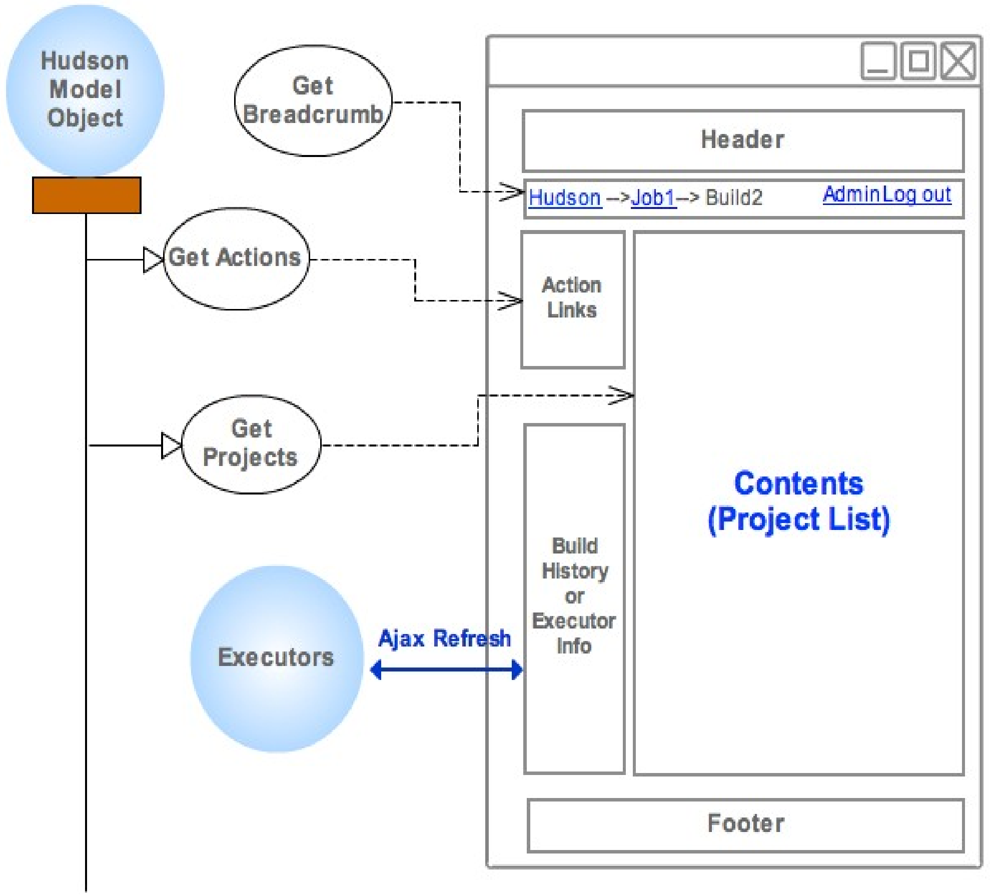
Since each Hudson page is related to a Model object, the page itself is rendered from the encapsulated data of the model object.

When a request for particular page reaches server, the server first tries to find the corresponding Model Object using stapler REST framework.

Next, server tries to finds the corresponding page definition file (Jelly File).

The tags defined in the page, use the methods of the objects to get rest of the information to render the full page.

Portion of the page can include segments for Ajax request. When XHR request is received by server, corresponding object is found in the same way as regular request and the HTML is rendered from the corresponding Jelly file.



Finding the Model Object

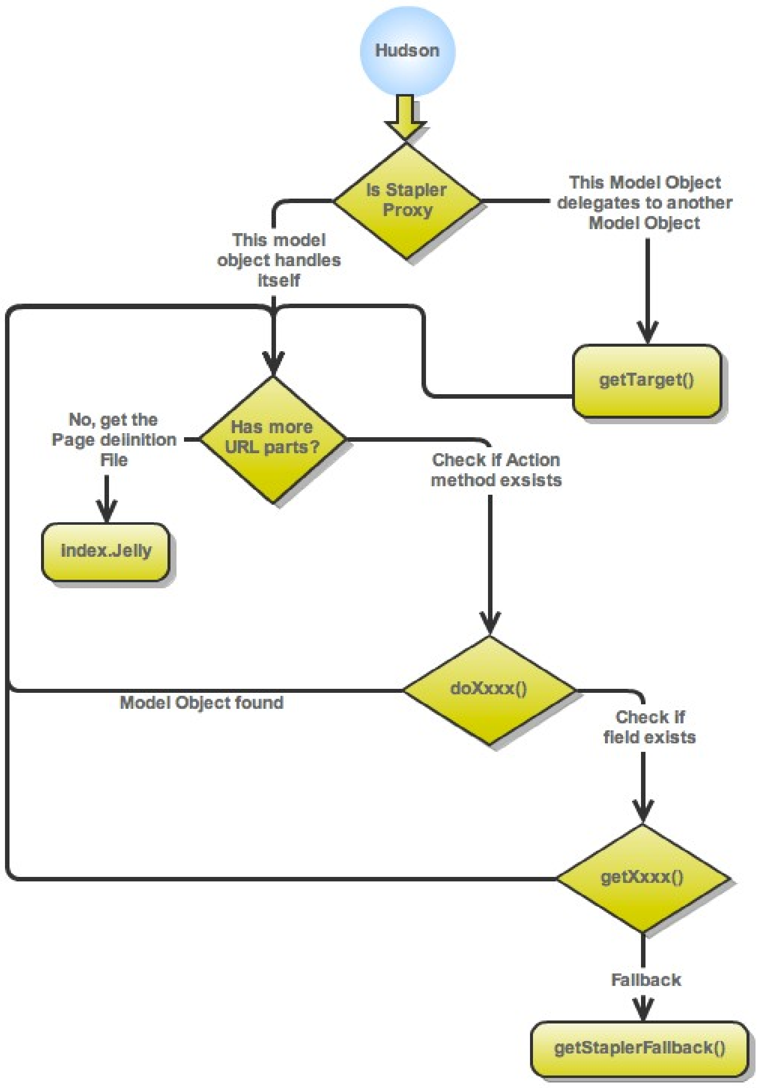
To find the Model Object, Hudson REST framework (stapler) applies a heuristic approach.

The search always starts with Singleton Model Object called Hudson. The following steps are applied

- Check if the model object extends StaplerProxy. If so call getTarget() to get the proxy object.
- If there are no more URL parts then that object is the model Object
- If has more URL parts, the evaluate for remaining portion of the URL.
- Apply the following steps
  - Next part may be an action, so try doXxx(...) method. Ex: if the URL portion is /configure then the method is doConfigure().
  - If action step fails, try to get corresponding field. Ex. If the URL portion is /job/2, then getJob(2) is tried.
- If all fails finally call the fallback method (getStaplerFallBack())

If the model object is found then find the Page Definition file corresponding to the Model Object.

If an object wants to control the URL mapping on its own, then must define a method called doDynamic() or getDynamic(). In case of doDynamic() it must consume entire remaining URL.



ORACLE

Finding the Jelly Files

Once the Model Object is found, server next tries to find the corresponding Page Definition File (Jelly File). They are the inputs to template engines used to render the HTML send to the client.

By convention Jelly files are placed as resources, organized by their class names. Also a Mode Object can provide a method getUrl() to provide the path of Jelly file.

Ex.

The model object *org.acme.foo.Bar* need to have Jelly class at */org/acme/foo/Bar/index.jelly*

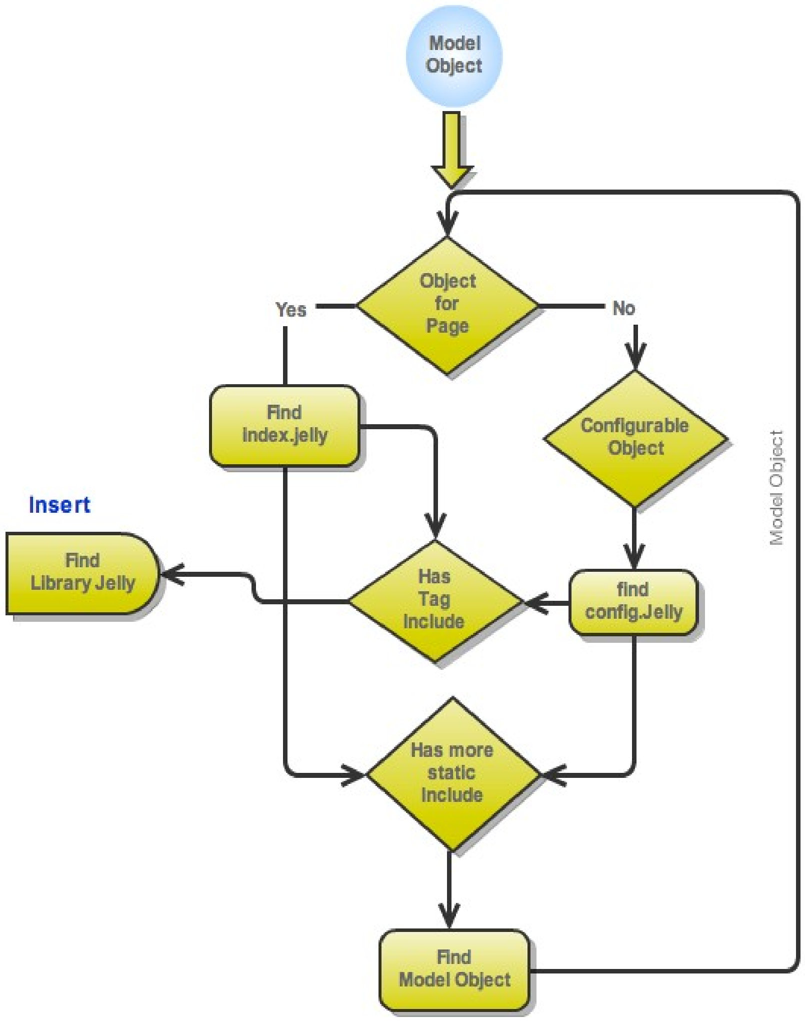
If the Jelly file has static include, then the Jelly (config or index) corresponding to the object is searched and included dynamically.

Ex. *<st:include from="${d}" page="${d.configPage}" optional="true" />*

Also the Jelly file can have tag based include. For Example

> *xmlns:l="/lib/layout"*
> *<l:main-panel>*
> *..*
> *</l:main-panel>*

In the above case *main-panel.jelly* from folder /lib/layout will be dynamically included.



Participation in View by Extensions

Extensions can participate in the views via a class that implements Describable interface. The Descriptor that describes the Describable object is responsible for configuring the view and saving and loading the corresponding metadata.

Plugins can provide two jelly file for configuration

global.jelly – For global configuration
config.jelly – For Extension Point configuration

Global Configuration

Descriptor.configure(req, json) is used to configure the describable object witl global config data, when the global configuration page is saved.

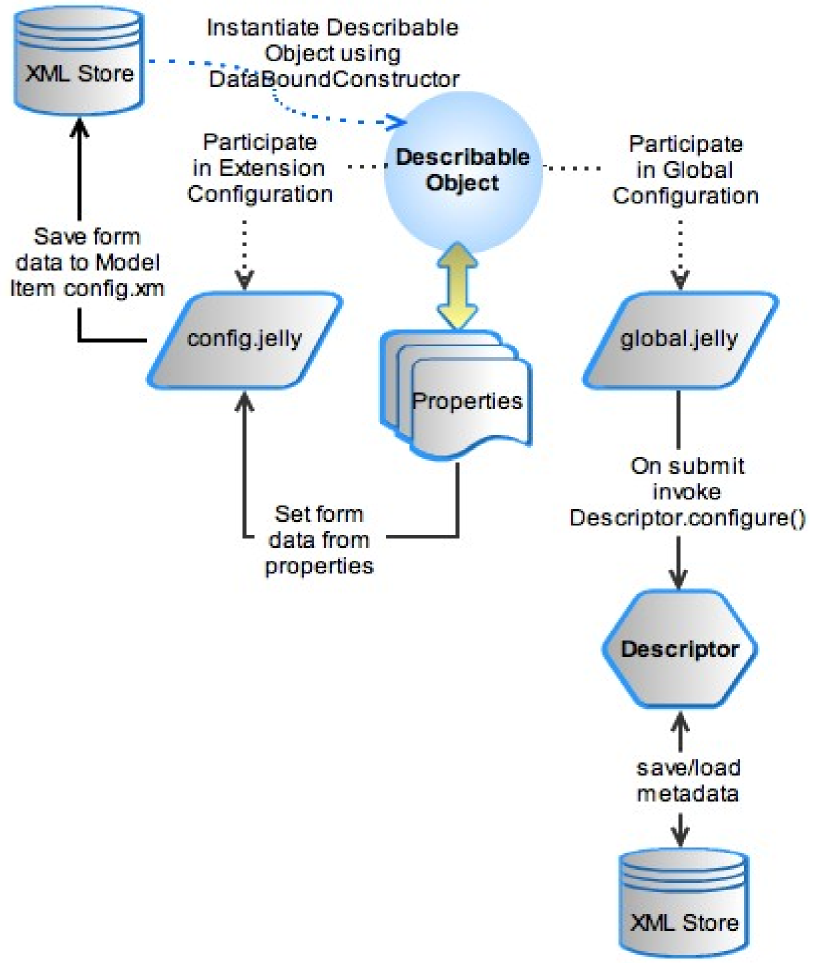
Descriptor.load/save() is used to load the global configuration data. The confi data is saved in its own XML file.

Extension Point Configuration

The view provided by config.jelly is included in the configuration Page of a Mo Item (ex. Job). The fields in the config.jelly is initialized with corresponding ge method in the Describable Object.

When the mode item configuration Page is saved, the field values of the corresponding Describable object is saved in the Model Item's config file. For example, the post-build configurations of a Job is saved in the Job's config fil

During Hudson initializes, the model items are constructed and correspondin Describable objects are created using the DataBoundConstructor (each constructor argument is equivalent to the form fields)


XML Store
Instantiate Describable Object using DataBoundConstructor
Describable Object
Participate in Extension Configuration
Participate in Global Configuration
Save form data to Model Item config.xm
config.jelly
Properties
global.jelly
Set form data from properties
On submit invoke Descriptor.configure()
Descriptor
save/load metadata
XML Store

Validating the Form Fields

The form values entered by user in configuration form can be validated. If validation fails, then error message set by the Extension will be displayed in the form.

The form validation occurs at various ways

- On the fly validation
- Action based validation

On the fly validation occurs at two stages. When the form is loaded, if the Form fields have validation URL, then they are called using XHR. The validation URL is specified as

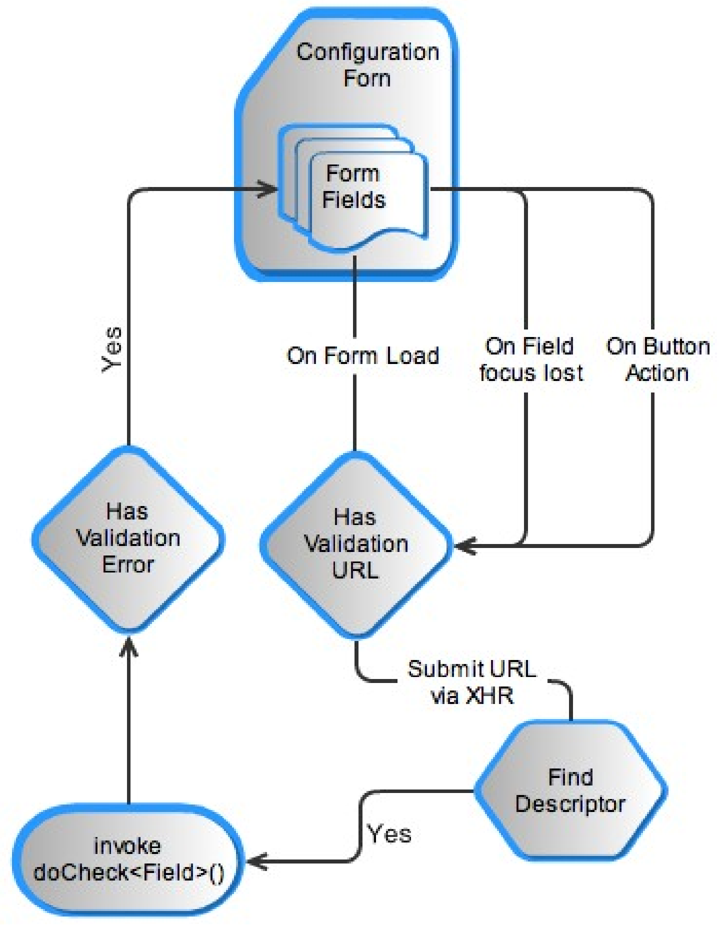
```
<f:textbox name="javadoc_dir" value="${instance.javadocDir}"
checkUrl="'descriptorByName/JavadocArchiver/check?
Value='+encodeURIComponent(this.value)"/>
```

The Descriptor of the Describable is searched in the Hudson Model hierarchy and doCheck() method is invoked on the Descriptor to validate the field. If error occurs, then it is displayed underneath the field.

Action based validation occurs if the ***validateButton*** defines a method for validation. Ex.

```
<f:validateButton title="${%TestConnection}" progress="${%Testing}"
method="testConnection" with="url,username,password" />
```

The method *testConnection* of the Descriptor is invoked via XHR and corresponding result is displayed in the field below the Button.


Configuration Form
Form Fields
Yes
On Form Load
On Field focus lost
On Button Action
Has Validation Error
Has Validation URL
Submit URL via XHR
Find Descriptor
Yes
invoke doCheck<Field>()