Creating a FormComponentAjaxBehavior

Wicket allows us to do almost anything that one can imagine, only require a little time how to do it. In the project wicket-stuff exists examples of integration with dojo, in particular this two links
DojoFeedbackIndicator and DojoValidationAjaxHandler call my attention because gives a nice feedback to the user about what's wrong in the form.
Now, one restriction that I found using the previous links is that I can't use it together (like I want), the validation error (alerticon) only works when I submit the form, no when an error ocurrs, and the indicator of error (show the input in red) disappears if I submi the form.

Overview
So, what I expect?
If I use a Form and declare a validator for a field, declare a "FormFieldValidator" associated to the field. If a validation error occur, change the "context" of the field (change the font color, for instance) , show an alerticon on the right of the Field and if I mouse the mouse over the image show me an message indicating which is the error; If the Form is submitted, the unique thing that whant to change is show an extra FeedbackPanel with all messages of the Form.

How to use it?
Simple, only add an behavior to the Field located in a Form, i.e.:

```
<field>.add(new FormFieldValidator(ClientEvent.BLUR));
```

Where ClientEvent.BLUR is the event when the validation is fired.
Extract:

```
...
public SignInForm(MarkupContainer parent, final String id)
                {
                        super(parent, id);
                        username = new TextField<Integer>(this, "username",
                                        new PropertyModel<Integer>(properties, "username"));
                        username.setRequired(true);
                        username.add(new PatternValidator("\\d+"));
                        username.add(new FormFieldValidator(ClientEvent.BLUR));
                        username.setOutputMarkupId(true);
...
}
```

Screenshots

Source code
Here is a quickstart with the package wicket.form.validation, that package contains the filed FormFieldValidator.java, alerticon.gif and ValidateFormField.js.

What's behind?
If you rech this part is that want to known how this works 😛.
First, at difference of the first two links used as reference, the code not depend of dojo, only extends AbstractDefaultAjaxBehavior and implements IHeaderContributor.
The next code it's inside FormFieldValidator.java

In the method onBind() verify that the component where I want the validation is a FormComponent instance, if not throw an exception.

```
....
        protected void onBind()
        {
                Component oComponent = getComponent();
                if (!(oComponent instanceof FormComponent))
                {
                        throw new WicketRuntimeException(
                                        "This handler must be bound to FormComponents");
                }

                this.formComponent = (FormComponent) oComponent;
                this.formComponent.add(new AttributeModifier("id", true, new Model<String>(
                                this.formComponent.getId())));
                this.formComponent.add(new AttributeModifier(sEventName, true, new Model()
                {

                        public java.lang.Object getObject()
                        {
                                return "javascript:"
                                                + "var wcall=wicketAjaxGet('"
                                                + getCallbackUrl()
                                                + "&"
                                                + formComponent.getId()
                                                + "=' + this.value, function() { }, function() { });return !
wcall;";
                        }
                }));
        }
....
```

Now, if the bind method not thrown an exception, when the ClientEvent is "fired" I execute a javascript function located in the file ValidateFormField.js called validate_form_field, but first validate the field to know if is valid or not.
The function has tree parameters:

1. **id** : identifier of the component
2. **type** : an string indicating if the field is 'valid' or 'invalid'
3. **message** : the string to show in the caption over the alerticon image. If the component is valid, the message is empty.
4. **alertImage** : path to the alerticon image.

```
@Override
        protected void respond(final AjaxRequestTarget oAjaxTarget)
        {
                formComponent.validate();
                if (!formComponent.isValid())
                {
                        final String sMessage = StringEscapeUtils.escapeJavaScript(formComponent
                                        .getFeedbackMessage().getMessage().toString());
                        oAjaxTarget.appendJavascript("validate_form_field('"
                                        + formComponent.getId() + "','invalid', '" + sMessage + "','"
                                        + formComponent.urlFor(ALERT_IMAGE).toString() + "')");
                }
                else
                {
                        oAjaxTarget.appendJavascript("validate_form_field('"
                                        + formComponent.getId() + "','valid', '')");
                }
        }
}
```

Here's the javascript function that do the "magic":

```
function validate_form_field(id,type,message,alertImage) {
    // Obtain the field that I want to validate.
    caja=document.getElementById(id);
    // the image input.jpg contains tree positions:
    //        top:normal.
    //        middle: focus.
    //        bottom: red image (error)

    // If the field is not valid
    if (type!='valid'){
        // put the image in the error position.
        caja.style.backgroundPosition='left bottom';
        // the font color to red.
        caja.style.color='red';
        // If the warning exists (the image), remove them. That'??s because I don'??t want //multiple images on
next errors.
        if(document.getElementById(id+'Warning') != null){
            caja.parentNode.removeChild(document.getElementById(id+'Warning'));
        }

        // Now create the warning (image and title).
        var img = document.createElement('img');
        img.height = img.width = 16;
        img.height = img.height = 16;
        img.src = alertImage;
        img.setAttribute('id',id+'Warning' );
        img.setAttribute('title',message);
        insertAfter(img,caja);
    }else{
        // If the field is valid, remove the warning.
        if(document.getElementById(id+'Warning') != null){
            caja.parentNode.removeChild(document.getElementById(id+'Warning'));
        }
        // Put the image in normal position.
        caja.style.backgroundPosition='left top'
        // Return the font color to black.
        caja.style.color='black';
    }
}

/*
 * This function only exists because jasvascript doesn't known about
 * insert anything after an element, only is valid insertBefore.
 */
function insertAfter(newElement,targetElement) {
        var parent = targetElement.parentNode;
        if(parent.lastchild == targetElement) {
                        parent.appendChild(newElement);
        } else {
                        parent.insertBefore(newElement, targetElement.nextSibling);
        }
}
```

The function validate_form_field contains some "extra" things.
One advantage of wicket is the concept of "Component" that allows us reuse code in different projects. But this apply in java code mostly. The look and feel is per project, so I want to accomplish the separation of concerns.

To do that only has to create a image like

that contains the position commented on the javascript and create a reference in the css file like

```
input { height:24px; border:0; padding:4px 6px 0 17px; width:120px; background: url(input.gif) left top no-
repeat ; }
input:hover { background-position:left center; }
input:focus { background-position:left center; }
```

The catch is that you has to create "many" of these that only differ in the length or height. This step is the unique thing "not reusable".
But if you don't want to spend time on this, don't do it. In the quickstart I show two ways, the plain without css and the "nice", so you can choose how many time want to dedicate to the L&F.

The code works on 2.0-SNAPSHOT (I don't use 1.x, so I don't known the API changes).