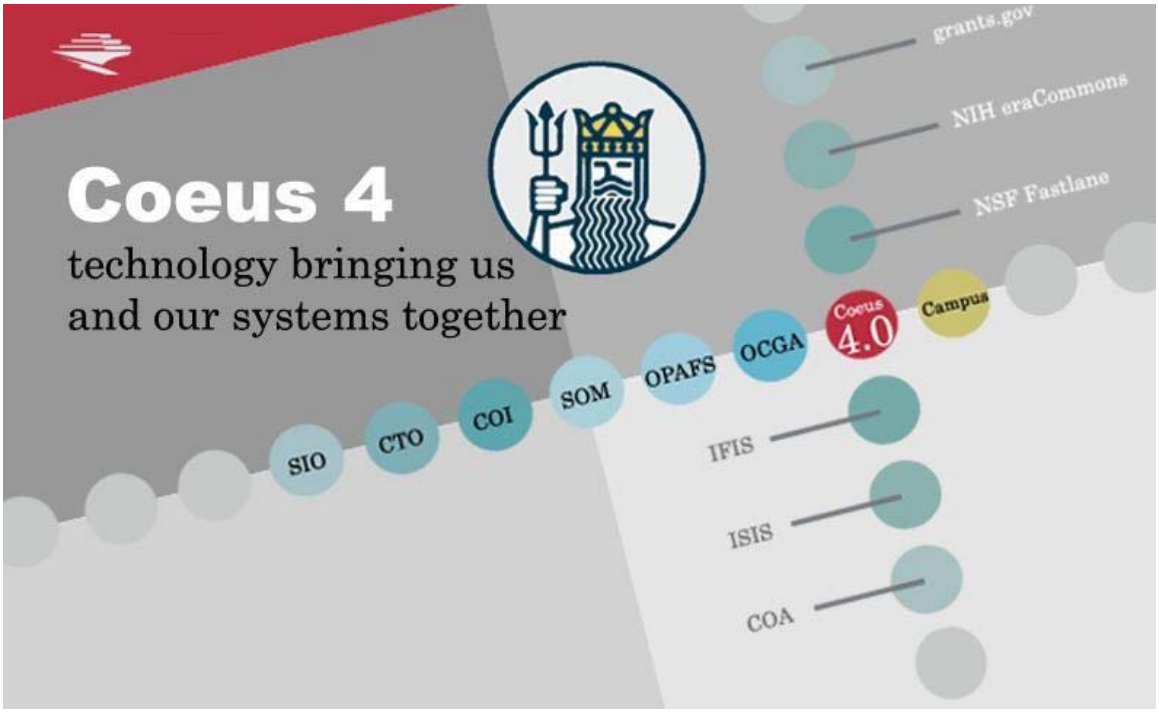
UCSD's Personalization Engine:

Local Customizations to COEUS



Coeus User Group Meeting
New Orleans, Louisiana
March 31, 2008

Presented by:

Mojgan Amini
maamini@ucsd.edu

Robert Dias
rdias@ucsd.edu

1 Personalization Engine – Overview

UCSD's Need

UCSD's decision to upgrade to the latest Coeus release, from the current v3.7 campus production version, spurred the need for many local customizations of the Coeus system. Three main areas of customization were identified:

- **Customization**: Localization of label names, placement of fields, additional tabs/screens, standardized look and feel.
- **Departmental Security**: restricting access to users based on roles and "rules".
- **Integration with other systems**: integration with UCSD's internal financial systems, as well as other systems.

Challenge

UCSD had the need to change the attributes and behavior of GUI components contained in all the forms based on the data being presented or entered. Some specific customizations include:

- Label and Tab text changes
- Placement of fields
- Adding/removing fields/screens
- Enabling/disabling buttons/functionality
- Color and attributes of fields and labels and screens
- Short-cut keys for functions
- Validation on fields/tabs

An additional goal was to make minimum changes to the base code.

Solution

The Personalization Engine was developed to handle the customizations without changing the base code. The personalization is achieved through an external XML configuration file located/created on the server.

2 Personalization Engine – How it Works

Three schemas, coeus_form schema, access_policy schema and authorization_form schema, handle this personalization. The coeus_form schema allows modification and addition of attributes, the access_policy schema allows setting access policy, and the authorization_form schema allows changes to the behavior of the form such as disabling menus, buttons, text fields and adding additional data at runtime. Two schemas are explained in detail later in this document. The access_policy schema is in the "proof of concept" phase.

Form Properties (setting field attributes)

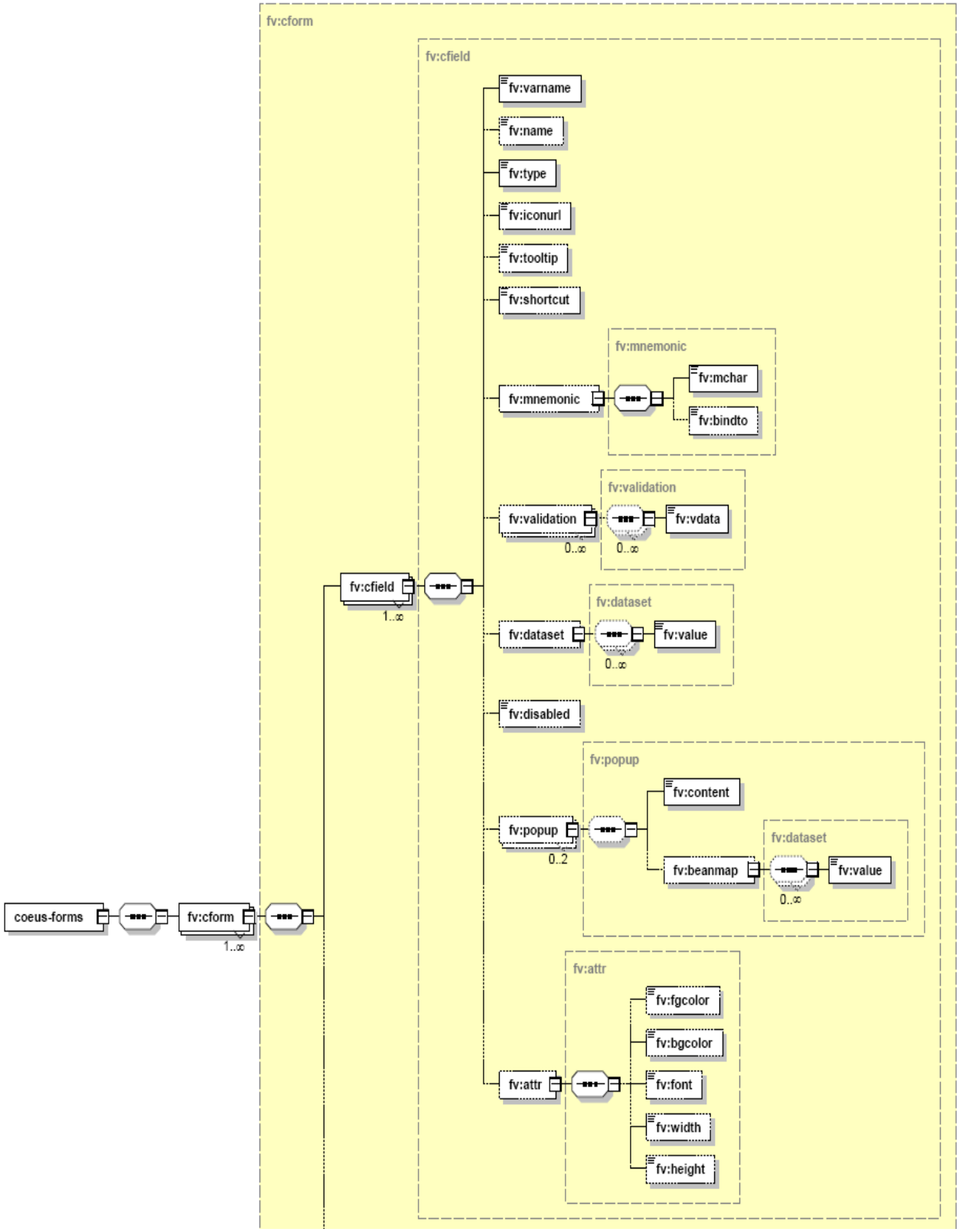
Changing Properties

The following is the properties (XML) file for setting attributes. In this example we are setting the label name for a field, changing the label color, assigning a short-cut key to a function, and validating input on a number range.

```
<cform classname="edu.mit.coeus.award.gui.AwardDetailForm">
        <cfield>
                <varname>lblAccount</varname>
                <name>Fund/Index:</name>
                <type>javax.swing.JLabel</type>
```

```
        </cfield>
        <cfield>
                <varname>lblSponsorAwardNo</varname>
                <type>javax.swing.JLabel</type>
                <attr>
                    <fgcolor>#FF0000</fgcolor>
                </attr>
        </cfield>
        <cfield>
                <varname>mnuItmModify</varname>
                <type>edu.mit.coeus.gui.menu.CoeusMenuItem</type>
                <shortcut>ctrl M</shortcut>
        </cfield>
        <cfield>
                 <!—This feature is not yet available →
                <varname>txtAuthorizedAmount</varname>
                <type>edu.mit.coeus.utils.DollarCurrencyTextField</type>
                <validation type="range">
                    <vdata datatype="low">0</vdata>
                    <vdata datatype="high">1000000</vdata>
                    <vdata datatype="errormesg">Authorized amount
exceeds specified limit</vdata>
                </validation>
        </cfield>
```

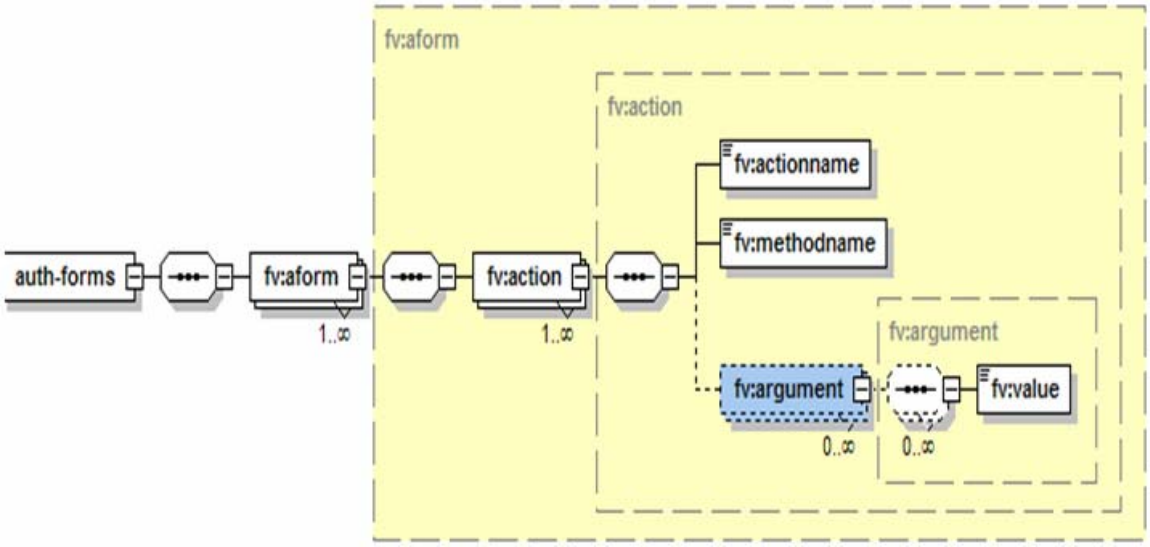
The following lists all the properties that can be customized, otherwise known as the XML schema.



Authorization Properties (setting access rights)
Authorization properties are used to restrict access to functions or fields based on a defined criterion. Here is how a sample policy XML file will look like for setting up access rules. This is in "proof of concept" phase.

```
<policy>
<rule id="ProposalList1">
  <unit>OCGA</unit>
  <role>updater</role>
   <bean type="proposal">
       <status>funded</status>
        …
        …
   </bean>
   <action target="button" form="xyz" name="abc">disable</action>
</rule>
 …..
</policy>
}
```

Authorization XML data feed is dynamically generated at run-time on the server. A form could have multiple actions for any given instance. Action has data associated with it to hold list, text, map, etc. which are basically action arguments. Actions are encapsulated into commands which are then invoked by the personalization controller in the client.



The following is an example of the generated XML data that is interpreted and displayed on the screen.

```
<aut:aform uniqueID="20063166" target="Proposal List" cache="true">
  <aut:action>
    <aut:actionname>Disable overwrite</aut:actionname>
    <aut:methodname>cmdDisableOverWrite</aut:methodname>
  </aut:action>
 </aut:aform>
…..
 <aut:aform uniqueID="20063179" target="Proposal List" cache="true">
  <aut:action>
    <aut:actionname>Enable overwrite</aut:actionname>
    <aut:methodname>cmdEnableOverWrite</aut:methodname>
  </aut:action>
```

3 Applications of Personalization Engine

In addition to changing labels, modifying attributes there are other features/add-ons that can be implemented via the personalization framework.

The personalization engine sits between the GUI renderer and the base coeus controller. This gives us the ability to personalize the results the way we want it. One neat application is context sensitive reporting. Currently MIT has implemented a Query Engine that allows us to query coeus data collection. This data is available in the form of a bean which can easily be converted to XML, and with the help of a style sheet can be formatted as desired. This report is context sensitive because it queries the active data collection that is been presented to the user. If additional data is needed that is not available it can easily be loaded from the database. In the personalization framework a window pop-up listener has been implemented which can be tied to any GUI component. The pop-up window can render this context-sensitive report.

Here is an example of how a pop-up is defined:

```
<coeus-forms xmlns="http://coeus.ucsd.edu/personalization/coeusforms">
- <cform classname="edu.mit.coeus.award.gui.AwardBaseWindow">
- <cfield>
  <varname>#new_field_summ</varname>
  <type>edu.mit.coeus.gui.toolbar.CoeusToolBarButton</type>
  <iconurl>http://localhost:8080/coeus/images/summary.gif</iconurl>
  <tooltip>Summary Panel</tooltip>
- <popup action="onclick" target="window" ctype="xsl">
  <content>http://localhost:8080/coeus/xsl/demo.xsl</content>
- <beanmap type="map" load="false">
  <value
keyindx="awardbean">edu.mit.coeus.award.bean.AwardDetailsBean</value>
  <value
keyindx="awardheader">edu.mit.coeus.award.bean.AwardHeaderBean</value>
  <value
keyindx="awardamount">edu.mit.coeus.award.bean.AwardAmountInfoBean</value>
  <value
keyindx="awardcmnts">edu.mit.coeus.award.bean.AwardCommentsBean</value>
  <value
keyindx="awardinvest">edu.mit.coeus.award.bean.AwardInvestigatorsBean</value>
  </beanmap>
```

```
        </popup>
        </cfield>
        </cform>
        </coeus-forms>
```

The beanmap defines the type of data you want in the style sheet.

Here is an example of a style sheet that is used for formatting the data.

```
<xsl:stylesheet xmlns:xsl='http://www.w3.org/1999/XSL/Transform' :coe="http://coeus.ucsd.edu/personalization/coeusbean"
version='1.0'>
<xsl:template match="/coe:coeus-beans">
<xsl:variable name="currentMitNumber">
<xsl:value-of select="coe:cbean[@beanid='awardheader']/coe:bean[@idx='0']/coe:beanval[@name='mitAwardNumber']"/>
</xsl:variable>
<html>
<body>
<h2><center>Award Summary Panel</center></h2>
<table border="1" cellpadding="2" width="100%">
<tr bgcolor="#151B8D" style="color: white; font-family: Verdana, Arial, Helvetica, sans-serif; font-weight: bold; font-size:
10px">
<th>UCSD Award #</th><th>Mod #</th><th>UCSD Proposal #</th><th>Sponsor Award #</th><th>Fund #</th>
<th>Start Date</th>          <th>End Date</th><th>Total</th><th>Comments</th>
</tr>
<tr bgcolor="#FFFFFF" style="color: black; font-family: Verdana, Arial, Helvetica, sans-serif; font-size: 10px">
<td><xsl:value-of select="$currentMitNumber"/></td>
<td><xsl:value-of
select="coe:cbean[@beanid='awardbean']/coe:bean[@idx='0']/coe:beanval[@name='modificationNumber']"/></td>
<td><xsl:value-of
select="coe:cbean[@beanid='awardheader']/coe:bean[@idx='0']/coe:beanval[@name='proposalNumber']"/></td>
<td><xsl:value-of
select="coe:cbean[@beanid='awardbean']/coe:bean[@idx='0']/coe:beanval[@name='sponsorAwardNumber']"/></td>
<td><xsl:value-of
select="coe:cbean[@beanid='awardamount']/coe:bean[@idx='0']/coe:beanval[@name='accountNumber']"/></td>
<td><xsl:value-of
select="coe:cbean[@beanid='awardbean']/coe:bean[@idx='0']/coe:beanval[@name='awardEffectiveDate']"/></td>
<td><xsl:value-of
select="coe:cbean[@beanid='awardamount']/coe:bean[@idx='0']/coe:beanval[@name='finalExpirationDate']"/></td>
<!-- Only show anticipated amount from parent -->
<td>
<xsl:for-each select="coe:cbean[@beanid='awardamount']/coe:bean[@idx]">
<xsl:if test="(coe:beanval[@name='rootMitAwardNumber'] = coe:beanval[@name='mitAwardNumber'])">
<xsl:value-of select="coe:beanval[@name='anticipatedTotalAmount']"/>
</xsl:if>
</xsl:for-each>
</td>
<!-- Only show general comments -->
<td>
<xsl:for-each select="coe:cbean[@beanid='awardcmnts']/coe:bean[@idx]">
<xsl:if test="(coe:beanval[@name='commentCode'] = '2')">
<xsl:value-of select="coe:beanval[@name='comments']"/>
</xsl:if>
</xsl:for-each>
</td>
</tr>
</table>
```

4 Integration into Coeus 4.3

The Personalization Engine will be included as part of the Coeus 4.3 release. It will come with basic sample schemas. The person responsible for Coeus implementation will need to add only the desired customizations to these configuration files, i.e. coeus_form.xml.

- Base Personalization Engine included in Coeus 4.3
  - Label, font, color changes on all components
  - Hover-over and URL links
  - Context-sensitive summary panels/reports
  - Pre-populating default values
  - Short-cut keys
- Extensions to Personalization Engine in future Coeus releases
  - Validation of input
  - Security Rules (based on data, user, status)

5 Summary

Leveraging off of the Personalization Engine, we are left in a good position to quickly make customizations without changing the base code, since the customizations are now achieved via changes to the properties files.

Moreover, we can extend the base functionality of Coeus by adding security-based rules in addition to setting UI properties.

As and added perk, we can now take updates to the base code with much less effort.