

Catalog Editor Demo  
Professor Gadia  
Com S 490  
Carl Chapman

### Rationale

Editing a text document by yourself is fairly straightforward, but creating a public, centrally important document like a University Catalog in an institution with many separate entities all having input is more complex. The main concerns revolve around communication and coordination. Is emailing documents back and forth an efficient solution? Professor Gadia had a better idea that revolves around a central Catalog document organized using XML.

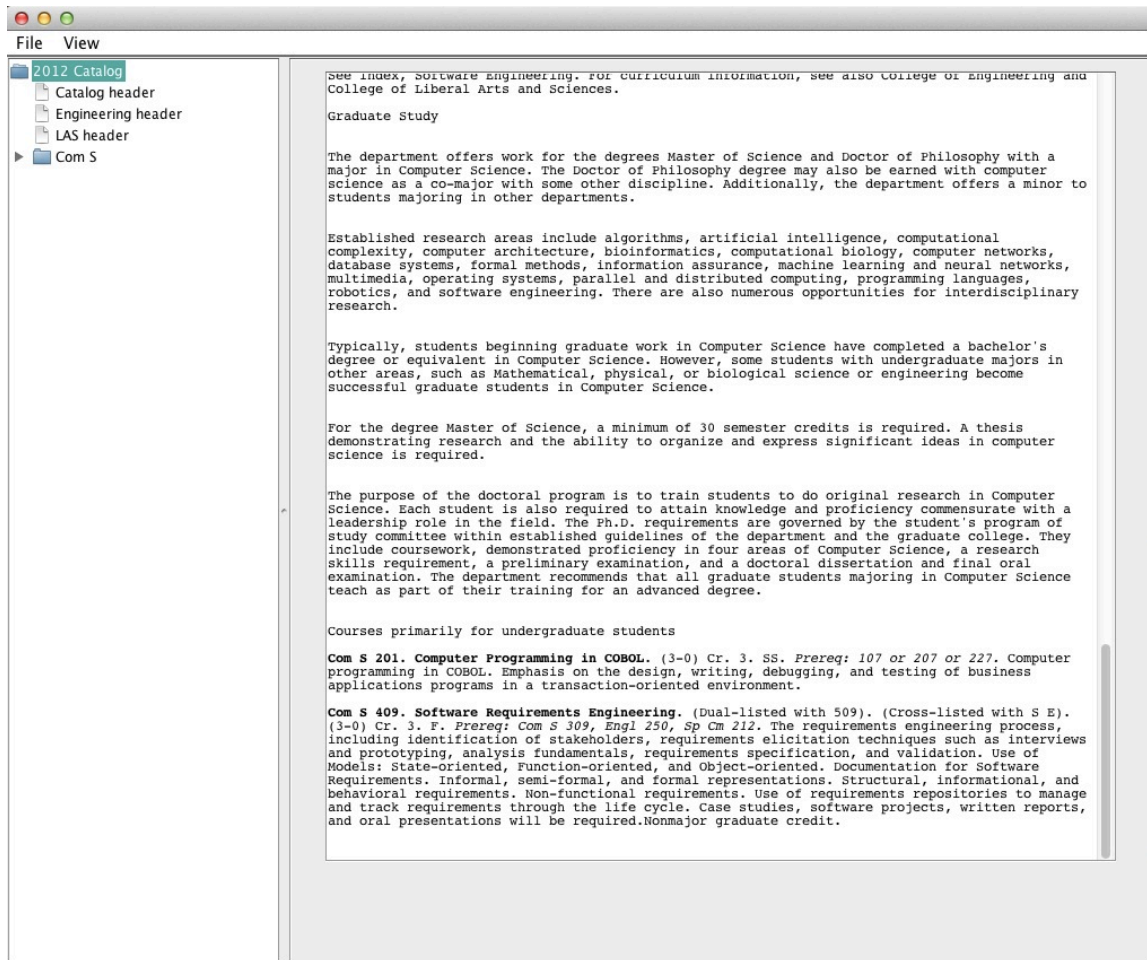
### Summary of Features

Unlike a relational database, modifications to the way the data is organized in an xml document can be done with minimal disruption. Because it uses xml to store everything, the Catalog document is queriable using Xpath from within the application. This is especially useful for administrating agents that need to prepare reports about the Catalog. A user can save queries they find useful and see what queries others have saved. A set of 'global' queries useful to people unfamiliar with Xpath is maintained by administrators. The queriable nature of the document can also simplify the work of services such as the iastate.edu website, schedule planning software and any service that periodically needs up-to-date Catalog related information.

Permissions management is built in so that a user can only modify areas of the document that they have been designated an editor of. The data is highly available, so that coordinating with someone else is as simple as looking at his or her latest version. Communication is as simple as writing a comment on that version. The GUI has various additional human-interaction-oriented sweeteners that are intended to make it convenient to use.

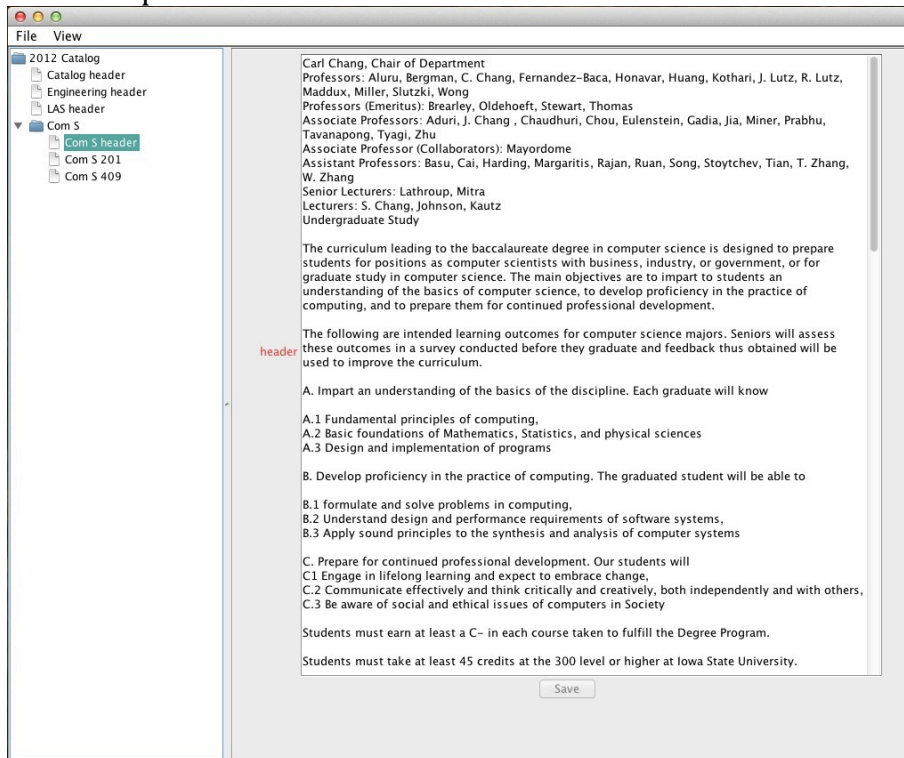
## Visual Tour

After opening the xml database document and validating credentials, the user is taken to the screen that they were last at. For new users, the default is the catalog root in the Editing view, pictured below:

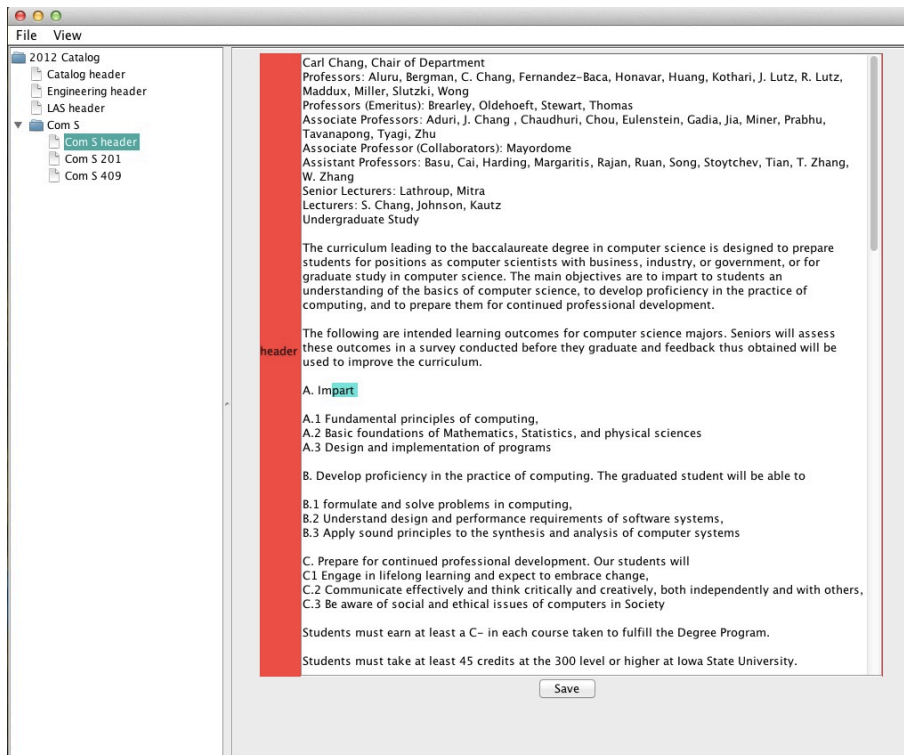


With the '2012 Catalog' node selected, the entire catalog is rendered in the right pane. Here a simplified version of the catalog is represented with a few headers and a single program (Com S) containing two Classes (201 and 409). I call this the static view. You can see the courses with their formatting, but you cannot edit them. The width of this window makes these courses look almost exactly like they do on the iastate.edu website. Styled text for the headers has not been implemented, so a few lines that should be bold in the header are normal text. A very similar static view of just one single program is also available the folder for a program is clicked. This sort of static view makes it easy to check for consistency between many courses at once, and is a familiar way to view the catalog.

To edit a header element, select header in the expanded tree, as shown on the left. The text will be in a scroll pane. Notice a faint red label on the left and a disabled save button.



When this text is out of sync with the doc, the label on the left becomes red and the save button becomes enabled. The same is true for several other views that use the same mechanics.



Saving or exactly replacing what was removed would bring the label colors and save button back to their previous state.

Similarly, the course version-editing page has similar color-coded alerts, but you will notice that the block text has some colors by default even when not out of sync. This color-coding is to help the user associate the part of the text block below with the field above.

The screenshot shows a web application for editing course information. On the left is a sidebar with a file tree containing '2012 Catalog', 'Catalog header', 'Engineering header', 'LAS header', and 'Com S'. The main area is a form for 'Software Requirements Engineering'. Fields include 'title', 'dual' (509), 'cross' (S E), 'credit' (3), 'primary' (3), 'secondary' (0), 'maxhours', 'prereq' (Com S 309, Engl 250, Sp Cm 212), 'description', and 'notes'. There are checkboxes for 'sfnonly', 'repreatable', 'nonmajorgc' (checked), and 'experimental'. Radio buttons are for 'F', 'S', 'SS' (S selected), 'none', 'odd', 'even', and 'all'. A 'Save' button is at the bottom. Below the form are two comment boxes: one with 'author: gadia' and text 'Description could be shortened.', and another with 'author: carl1978' and text 'What about adding a lab element to the course?'. A plus button is at the bottom right.

You might notice the comments below. The entire panel is in a scroll pane and you can just barely see that the '+' button is a little bit off the page. New comments get inserted above the '+' button and make panel longer. I felt that this was preferable to having all the comments stuffed in a little scroll pane. Clicking the plus button if you already have a comment causes your existing comment to blink a darker grey, as I have tried to capture in the above screenshot. This is to prompt the user to edit their comment instead of adding more and more comments.

The text that has a white background in the block cannot be edited, as it is standardized. Notice the 'Nonmajor graduate credit.' at the bottom corresponds to a checkbox - that is an example of a boolean there-or-not-there kind of field.

Below I have altered some of the fields to show off the colors that change.

title	Software Requirements Engineering For Noobs		
dual	509	cross	S E, Cpr E
credit	3	<input checked="" type="checkbox"/> sfonly	F <input type="radio"/> S <input type="radio"/> SS <input type="radio"/> none <input type="radio"/> odd <input type="radio"/> even <input type="radio"/> all <input checked="" type="radio"/>
primary	3	<input checked="" type="checkbox"/> repeatable	
secondary	1	<input checked="" type="checkbox"/> nonmajorgc	
maxhours	7	<input checked="" type="checkbox"/> experimental	
prereq	Com S 309, Engl 250, Sp Cm 212		
description	The requirements engineering process, including identification of stakeholders, requirements elicitation techniques such as interviews and prototyping, analysis fundamentals, requirements specification, and validation. Use of Models: State-oriented, Function-oriented, and Object-oriented. Documentation for Software Requirements. Informal, semi-formal, and formal representations. Structural, informational, and behavioral requirements. Non-functional requirements. Use of requirements repositories to manage and track requirements through the life cycle. Case studies, software projects, written reports, and oral presentations will be require.		
notes	You will Like this course, trust me.		
Com S 409X. Software Requirements Engineering For Noobs. (Dual-listed with 509). (Cross-listed with S E, Cpr E). (3-1) Cr. 3. Repeatable. F. S. SS. Prereq: Com S 309, Engl 250, Sp Cm 212. No more than 7 credits of Com S 409X may be counted toward graduation. The requirements engineering process, including identification of stakeholders, requirements elicitation techniques such as interviews and prototyping, analysis fundamentals, requirements specification, and validation. Use of Models: State-oriented, Function-oriented, and Object-oriented. Documentation for Software Requirements. Informal, semi-formal, and formal representations. Structural, informational, and behavioral requirements. Non-functional requirements. Use of requirements repositories to manage and track requirements through the life cycle. Case studies, software projects, written reports, and oral presentations will be require. Nonmajor graduate credit. Satisfactory-fail only. You will Like this course, trust me.			
Save			
author: gadia	Description could be shortened.		
author: carl1978	What about adding a lab element to the course?		
+			

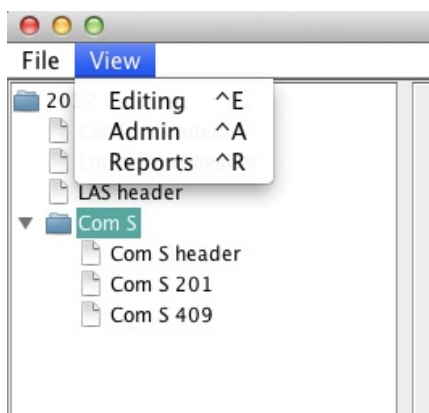
My intention in doing all of this color-coding was to make sure that when editing the text block, you can easily figure out which field you are editing. This marginalizes the possibility of accidentally adding text meant for the description into the prereqs or notes fields, for example. It also makes the transition back to a saved state more dramatic, so users are always clear about if their data is transient or not. If you navigate away from this view in this state, your modifications will be discarded.

Notice on the third line down, how the 'X' suffix has been added to the text in 'No more than 7 credits of Com S 409X...' simply because the experimental checkbox is now selected. Deleting the 'X' next to the course designator in the upper left of the text block unchecks the checkbox and removes the X from the ...credits of... part of the text block. Below is the same content after I have deleted the 'X' and saved.

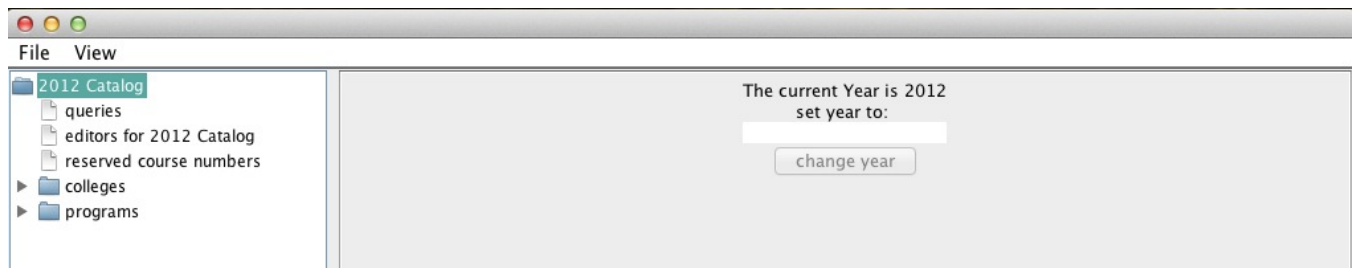


title	Software Requirements Engineering For Noobs		
dual	509	cross	S E, Cpr E
credit	3	<input checked="" type="checkbox"/> sfonly	
primary	3	<input checked="" type="checkbox"/> repeatable	F S SS none
secondary	1	<input checked="" type="checkbox"/> nonmajorgc	odd even
maxhours	7	<input type="checkbox"/> experimental	all
prereq	Com S 309, Engl 250, Sp Cm 212		
description	<p>The requirements engineering process, including identification of stakeholders, requirements elicitation techniques such as interviews and prototyping, analysis fundamentals, requirements specification, and validation. Use of Models: State-oriented, Function-oriented, and Object-oriented. Documentation for Software Requirements. Informal, semi-formal, and formal representations. Structural, informational, and behavioral requirements. Non-functional requirements. Use of requirements repositories to manage and track requirements through the life cycle. Case studies, software projects, written reports, and oral presentations will be require.</p>		
notes	You will Like this course, trust me.		
<p>Com S 409. <b>Software Requirements Engineering For Noobs</b>. (Dual-listed with 509). (Cross-listed with S E, Cpr E). (3-1) Cr. 3. Repeatable. F. S. SS. Prereq: Com S 309, Engl 250, Sp Cm 212. No more than 7 credits of Com S 409 may be counted toward graduation. The requirements engineering process, including identification of stakeholders, requirements elicitation techniques such as interviews and prototyping, analysis fundamentals, requirements specification, and validation. Use of Models: State-oriented, Function-oriented, and Object-oriented. Documentation for Software Requirements. Informal, semi-formal, and formal representations. Structural, informational, and behavioral requirements. Non-functional requirements. Use of requirements repositories to manage and track requirements through the life cycle. Case studies, software projects, written reports, and oral presentations will be require. Nonmajor graduate credit. Satisfactory-fail only. You will Like this course, trust me.</p>			
Save			

For admin users, there are three views: the editing, admin and reports views. Here is a screenshot of the menu for an admin user. For a non-admin user, the 'Admin' menu item is disabled.



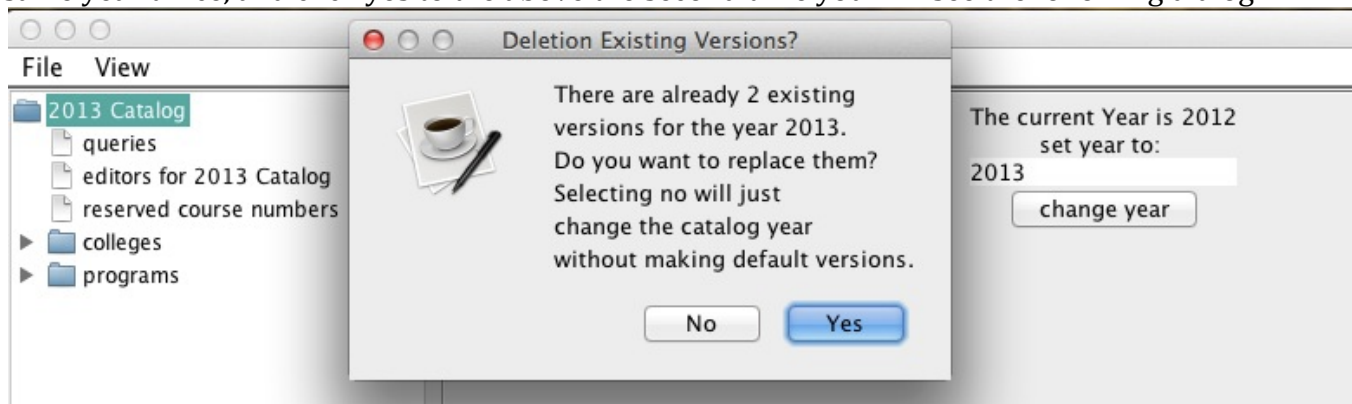
Selecting the Admin view returns the user to the last view they were at, or if they have never been in the admin view before, it defaults to the root catalog element. The admin view when root is selected is a widget that allows the admin to set the year. Eventually the database 'currentYear' variable needs to be updated, and though it could be done automatically, there are some choices that need to be made. Below is the view of the year setting widget:



The text field filters out all non-digits, though a full validity check has not been implemented. This should probably be changed to a combo box, but for a demo the text field works well enough. Entering numbers and pressing change year brings up the following dialog:



This gives you a chance to back out. Right now the logic creates default versions for every course, though it should probably clone old versions and set their 'catalogYear' to the new 'currentYear'. The basic idea is that there is only one version of a course per year, so if you set the year to the same year twice, and click yes to the above the second time you will see the following dialog:



This allows you to change the catalog year without making a bunch of default versions that replace existing versions.

Next the queries admin view allows management of global queries.



Every query must have a brief description, visible as a tool tip as shown above in the small brown box near the bottom center of the picture.

The green outline around the top two queries indicates that they are set as global. This setting just means that some admin thinks that this query is likely to be widely used. This could be supported by statistical analysis of use, or just by the intuition of an admin. Pressing the little circular image of a globe on the right side toggles this setting with a confirmation dialog. The red 'X' removes the query from the Database entirely, also with a confirmation dialog.

The green play button runs the query. Because I am expecting hundreds of queries to appear in this screen, the results pop up in a dialog, and the root element is a new element called 'RESULTS' that has an attribute for query that contains the query. The dialog's title is the query's description. Below is an example of the middle query's dialog popup.



Any number of queries can be added to the list of queries by filling out the top two text fields and pressing the '+' button, as expected.

Editing permissions are navigated by expanding the tree on the left to get to the desired section:

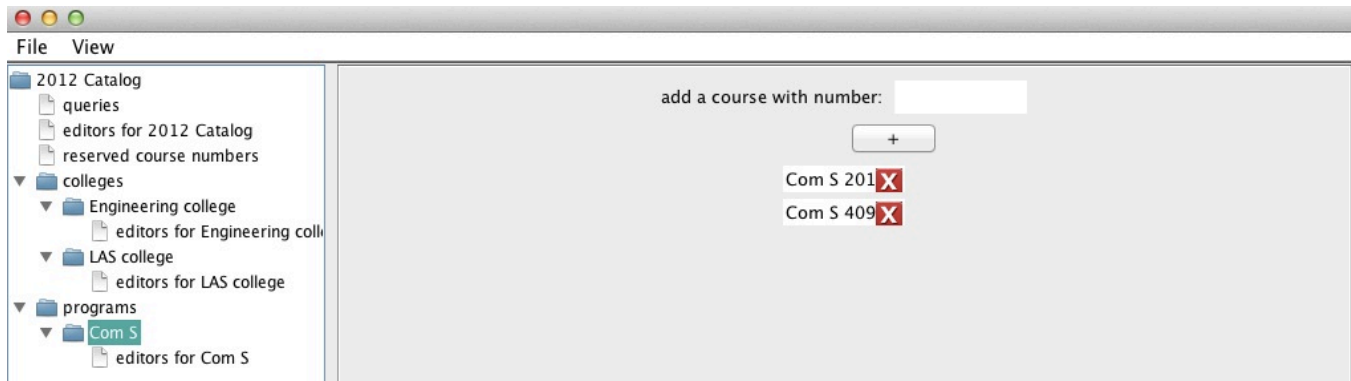


Editors can be added by writing their netID and pressing the '+' button, and removed with the 'x'.

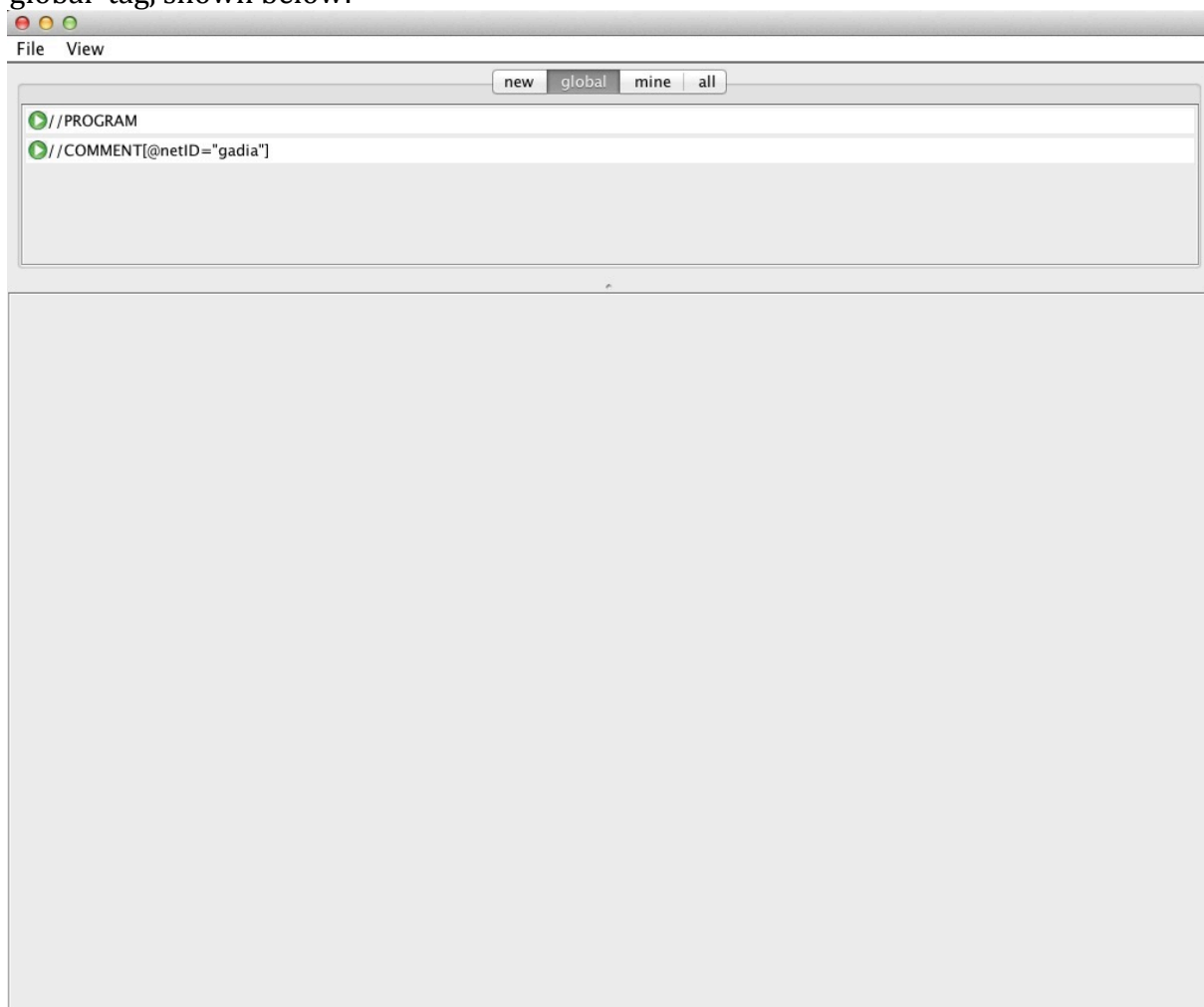
Editing the reserved course numbers looks very similar to editing a header text block except it is a smaller text area. It could be moved into the catalog root view.



Adding or removing a course can be managed by selecting a program node. The same conventions with the '+' and 'x' apply. Removal is always preceded by a confirmation dialog.



For the reports view, the screen shown is always the last tab selected, and the default is the 'global' tag, shown below:



The 'mine' and 'all' tabs display queries that the current editor is author of, and all saved queries respectively. A user can delete their own queries, so there is a red 'X' for queries on that page.


The 'new' tab is where a user can write their own queries, or execute queries without saving them.

File View

new global mine all

write an XPath query:

brief description:

test the current query  ☐ display in separate window save new query to 'mine'


There is an option to display the results in a separate window, but the default is to display the results in the large area below. This setting is saved so that a user can set it once and it will stay that way. As an example of making a new query, here are all the 2012 course versions:

File View

new global mine all

write an XPath query:

brief description:

test the current query  ☐ display in separate window save new query to 'mine'

```
<RESULTS query="//VERSION[@catalogYear='2012']">
<VERSION catalogYear="2012" experimental="false" sfOnly="false">
<TITLE>Computer Programming in COBOL</TITLE>
<DUAL />
<CROSS />
<CONTACTHOURS>
<PRIMARY>3</PRIMARY>
<SECONDARY>0</SECONDARY>
</CONTACTHOURS>
<CREDIT repeatable="false" maxCreditCount="">3</CREDIT>
<OFFERED>
<TERM yearsOffered="all">SS</TERM>
</OFFERED>
<PREREQ>107 or 207 or 227</PREREQ>
<DESCRIPTION>Computer programming in COBOL. Emphasis on the design, writing, debugging, and testing of business applications programs in a transa
<NOTES nonMajorGraduateCredit="false" />
<COMMENTS>
<COMMENT netID="carl1978">This comment is an example</COMMENT>
</COMMENTS>
</VERSION>
<VERSION versionID="3" authorID="carl1978" creationDate="12/12/2010" lastEdited="01/01/2011" catalogYear="2012" accepted="true" experimental="fa
<TITLE>Software Requirements Engineering For Noobs</TITLE>
<DUAL>509</DUAL>
<CROSS>S E, Cpr E</CROSS>
<CONTACTHOURS>
<PRIMARY>3</PRIMARY>
<SECONDARY>1</SECONDARY>
</CONTACTHOURS>
<CREDIT repeatable="true" maxCreditCount="7">3</CREDIT>
<OFFERED>
<TERM yearsOffered="all">F</TERM>
<TERM yearsOffered="all">S</TERM>
<TERM yearsOffered="all">SS</TERM>
</OFFERED>
<PREREQ>Com S 309, Engl 250, Sp Cm 212</PREREQ>
<DESCRIPTION>The requirements engineering process, including identification of stakeholders, requirements elicitation techniques such as interviews an
<NOTES nonMajorGraduateCredit="true">You will like this course, trust me.</NOTES>
```

## Implementation

Most of the complexity of this program comes from the editable block of text representing a version of a course. Parts of the text should never be edited and their presence or absence depends on editable parts. The text stored in the xml node is not always the same as what is displayed. The schema could be flattened to make implementation simpler, but then the document's usefulness as a queriable document with a natural organization would be degraded. The following xml snippet represents the xml schema for a version of a course that was arrived at after several lengthy discussions, including a consultation with Char from the registrar's office.

```
<VERSION versionID="3" authorID="carl1978" creationDate="12/12/2010"
lastEdited="01/01/2011" catalogYear="2012" accepted="true" experimental="false"
sfOnly="false">
  <TITLE>Software Requirements Engineering</TITLE>
  <DUAL>509</DUAL>
  <CROSS>S E</CROSS>
  <CONTACTHOURS>
    <PRIMARY>3</PRIMARY>
    <SECONDARY>0</SECONDARY>
  </CONTACTHOURS>
  <CREDIT repeatable="false" maxCreditCount="">3</CREDIT>
  <OFFERED>
    <TERM yearsOffered="all">F</TERM>
  </OFFERED>
  <PREREQ>Com S 309, Engl 250, Sp Cm 212</PREREQ>
  <DESCRIPTION>The requirements engineering process, including identification of
stakeholders, requirements elicitation techniques such as interviews and prototyping, analysis
fundamentals, requirements specification, and validation. Use of Models: State-oriented, Function-
oriented, and Object-oriented. Documentation for Software Requirements. Informal, semi-formal,
and formal representations. Structural, informational, and behavioral requirements. Non-
functional requirements. Use of requirements repositories to manage and track requirements
through the life cycle. Case studies, software projects, written reports, and oral presentations will
be required.</DESCRIPTION>
  <NOTES nonMajorGraduateCredit="true" />
  <COMMENTS>
    <COMMENT netID="gadia">This schema might change, but looks pretty good for
now.</COMMENT>
    <COMMENT netID="carl1978">A compromise between many tugging forces.</COMMENT>
  </COMMENTS>
</VERSION>
```

What is the best way to translate the above schema into a manageable system?

The following list of 17 constants best articulates the abstractions that solved the problem, and their type. Here, a 'sandwich' is something where the center is editable but the edges are always the same and shouldn't be edited.

```
EXPERIMENTAL = 0; // boolean attribute
TITLE = 1; // string element
DUAL = 2; // sandwich element
CROSS = 3; // sandwich element
```

```
PRIMARY = 4; // string element
SECONDARY = 5; // string element
CREDIT = 6; // string element
REPEATABLE = 7; // boolean attribute
MAXHOURS = 8; // custom sandwich element
FALL = 9; // custom object
SPRING = 10; // custom object
SUMMER = 11; // custom object
PREREQ = 12; // sandwich element
DESCRIPTION = 13; // string element
NONMAJORGC = 14; // boolean attribute
SFONLY = 15; // boolean attribute
NOTES = 16; // string element
```

There are several levels of information: what is stored on the file, what the state of the in-memory jdom2 xml document is, and what the user sees in the GUI. In order to implement a fully editable Course version while also implementing the formatted text block, I needed an additional abstraction layer. The text block just can't provide complete access to the above 17 variables if some of them are absent and the text block is supposed to look like it will in the catalog. This is why the 'field' view is so important.

The principal abstraction is that of a facade for each tracked concept that always stays consistent with what the GUI should display. For the editor, there are two interfaces provided to the user that they can edit: the block of text rendered to appear similar to the way it should appear in the Catalog, and a set of text fields, checkboxes and radio buttons that can also be used to control content. Making a change in one interface instantly changes the other.

There is a save button that is enabled only if the façade does not match the org.jdom2.Document (doc), which is always kept consistent with the underlying xml file. Fields and text background are color coded so that the user can more easily recognize the connection between the two interfaces. If a particular field is out of sync with the doc, it will be much more brightly colored until it either goes back to being in sync, or the document is saved (thus going back to being in sync).

## Challenges

I was first challenged by trying to use the DOM functions built into Java, and eventually found that jdom2 was best suited for this project and many coding tasks became much more straightforward. Xpath with jdom2 caused some exceptions until the jaxen library was added. It took a lot of false starts before I realized that I couldn't find a way to alter a javax.swing.text.Element (also, note the confusion with both jdom2 and swing making central use of the terms 'Element' and 'Document'...). The solution I came up with was to completely replace the entire block of text with each modification. For a paragraph-sized block of text, the performance lag is almost unperceivable.

The next challenge was what I'd call 'listener echo'. There is no problem for the text fields whose setText() methods seem to be atomic, but altering the block requires removing and then re-inserting. There are 18 individual listeners that are all triggered by a remove, so they all modify their own facades, which then try to modify the text fields, which then re-notify the façade which then tries to modify the block segments, who are about to re-insert and this echo always causes the program to crash. The solution I came up with was to remove the block listeners before the block removes and re-inserts, and then put them back on after it is done. This worked much better, but using a standard ArrayList - type listener list for the façade resulted in a doubling of listeners registered with each modification. After 30 modifications or so, the JVM was out of

memory! Clearly the listeners weren't all being removed before a modification, so some echo was causing extra listeners to be added during overlapping insertions.

I wrestled with this for a while, at first trying to remove all listeners of a certain type with each remove. That helped but it wasn't great. For this project, a simple array of size two is always a big enough listener list, so there is no need for an ArrayList, so I switched to the small array. Even with this modification, there was still some creepy thread behavior, where swing's single Event Dispatch Thread (EDT) was not doing things in the order I would expect.

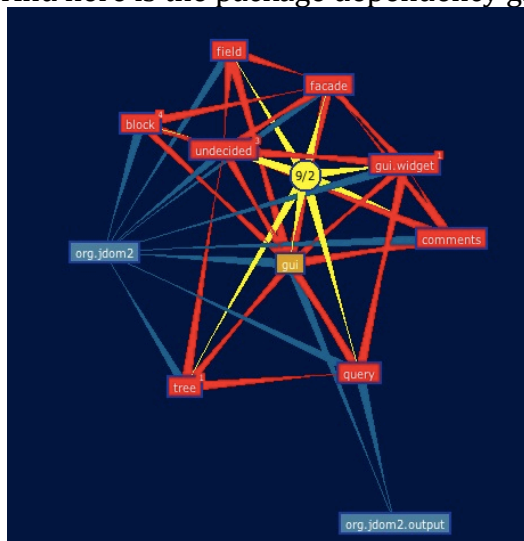
The best solution I could come up with was to use 'SwingUtilities.invokeLater()' which (as I understand it) allegedly waits for the EDT to finish and then blocks it while executing. Using this utility to remove all the listeners before doing things that would trigger them greatly improved performance and that's the way the application works now. However, every once in a while the two GUI representations get out of sync and stop working correctly. A simple switching of views resets the GUI to a good state. From debugging, it seems that every once in a while the removing thread finishes after the inserting thread so that some of the block listeners are not registered because they were not removed until after the insertion was complete. This results in an unresponsive block and I'm still pondering the best way to fix it. It almost never happens, so I am leaving things the way they are for now. (Suggestions are welcome - check out the source: <https://github.com/softwarekitty/CatalogEditorDemo> )

## Metrics

I used a metrics plug-in to evaluate my code. Here are the results without any modifications:

Metric	Total	Mean	Std. Dev.	Maximum	Resource causing Maximum	Method
► Number of Overridden Methods (avg/max per type)	23	0.365	0.803	5	/CatalogEditorDemo/src/facade/OfferingFacade.java	
► Number of Attributes (avg/max per type)	127	2.016	1.931	8	/CatalogEditorDemo/src/query/QueryControlPane.java	
► Number of Children (avg/max per type)	14	0.222	0.629	3	/CatalogEditorDemo/src/facade/AbstractFacade.java	
► Number of Classes (avg/max per packageFragment)	63	7	2.906	13	/CatalogEditorDemo/src/block	
► Method Lines of Code (avg/max per method)	2450	7.632	11.468	90	/CatalogEditorDemo/src/gui/StaticDocument.java	getVSpecsFromElement
► Number of Methods (avg/max per type)	282	4.476	3.817	17	/CatalogEditorDemo/src/block/VersionDocument.java	
► Nested Block Depth (avg/max per method)		1.421	0.786	5	/CatalogEditorDemo/src/gui/CatalogSettingsPane.java	actionPerformed
► Depth of Inheritance Tree (avg/max per type)		3.46	1.744	6	/CatalogEditorDemo/src/gui/Main.java	
► Number of Packages	9					
► Afferent Coupling (avg/max per packageFragment)		10.778	9.089	24	/CatalogEditorDemo/src/gui	
► Number of Interfaces (avg/max per packageFragment)	6	0.667	1.054	3	/CatalogEditorDemo/src/facade	
► McCabe Cyclomatic Complexity (avg/max per method)		1.988	2.175	18	/CatalogEditorDemo/src/undecided/Util.java	getNameFromID
► Total Lines of Code	4115					
► Instability (avg/max per packageFragment)		0.45	0.207	0.667	/CatalogEditorDemo/src/block	
► Number of Parameters (avg/max per method)		1.053	1.192	6	/CatalogEditorDemo/src/facade/SandwichFacade.java	SandwichFacade
► Lack of Cohesion of Methods (avg/max per type)		0.234	0.297	0.803	/CatalogEditorDemo/src/facade/AbstractFacade.java	
► Efferent Coupling (avg/max per packageFragment)		5.333	2.309	10	/CatalogEditorDemo/src/gui	
► Number of Static Methods (avg/max per type)	39	0.619	3.739	28	/CatalogEditorDemo/src/undecided/Util.java	
► Normalized Distance (avg/max per packageFragment)		0.421	0.242	0.87	/CatalogEditorDemo/src/undecided	
► Abstractness (avg/max per packageFragment)		0.129	0.13	0.364	/CatalogEditorDemo/src/facade	
► Specialization Index (avg/max per type)		0.219	0.519	2.667	/CatalogEditorDemo/src/block/VersionEditorKit.java	
► Weighted methods per Class (avg/max per type)	638	10.127	12.791	81	/CatalogEditorDemo/src/undecided/Util.java	
► Number of Static Attributes (avg/max per type)	44	0.698	2.798	18	/CatalogEditorDemo/src/facade/AbstractFacade.java	

And here is the package dependency graph:



Clearly it could use some refactoring. ☺