2014

An XML-based framework for management of a course catalog system with zero information-loss

Xiaofeng Wang
Iowa State University

An XML-based framework for management of a course catalog system with zero information-loss

by

Xiaofeng Wang

A thesis submitted to the graduate faculty
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Science

Program of Study Committee:
Shashi K. Gadia
Leslie Miller
Wallapak Tavanapong

Iowa State University
Ames, Iowa
2014

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

The transition from paper-oriented record keeping to their on-line counterpart has been anything but smooth. Often, the on-line systems are difficult to use and do not provide a structure to maintain organizational memory and procedures. A user momentarily sees the part he / she is currently working on and not aware of the big picture. Often, the user is has to work in such convoluted ways that require considerable learning curve. The Course Catalog at universities seem to be facing this these problems.

In this thesis we provide XCCat, an XML-based road-map for an enterprise solution for design, implementation, usage, and maintenance of course catalogs for institutions of higher learning. XCCat brings information about curriculum requirements, courses, changes to courses, approval sequences, participants and deliberations surrounding catalog development, and report generation under a single umbrella in a live XML-based database that not only remembers everything but can also be queried for this memory. The concept of catalog publication is carefully developed so when a catalog for the next year is published, the development version of the catalog for the following year is also published; the latter automatically carries forward the unfinished work in progress. This simplifies many things: need for tracking is reduced considerably, the development version of the catalog is always available, increases opportunities for collaboration, there is only one goto place foll all participants (e.g. university, colleges, and programs) for all catalog related issues. There is only one view of the catalog, all information is visible to everyone except the parts that need authorization for making updates. A fair portion that is enough to assess the viability of the ideas behind XCCat have already been implemented.

The approach comes close to what has been termed zero information-loss in the field of temporal databases. In the zero information loss model, one could query data, the database history of data, query updates, and query queries. Zero information-loss is a part of what is known as provenance in databases and information systems these days.

CHAPTER 1. INTRODUCTION

Development of a course catalog is a continual undertaking at institutions of higher learning. In old days one used paper-based systems and their automation has not been smooth. In some ways the new systems have become more complex to interact with, unforeseen problems have surfaced, and one is left feeling that we have not fully harnessed the advantages offered by computer and connectivity technologies. The reasons for this shortcoming are primarily rooted in the choice of technology platforms on which the solutions are being built. We envision that Extensible Markup Language (XML) – coupled with its associated technologies such as DOM [4], XPath [2], and XQuery [3] – provide a viable approach. In this thesis we report *XML-based Catalog Management System* (*XCCat – read axicat*), an enterprise solution for management of course catalog. Although we concentrate on the course catalog at Iowa State University, it is a sufficiently detailed use case that should extend to most institution of higher learning.

The prevalent legacy technologies, such as SQL-based relational databases and spreadsheets are inapt for hosting many applications for which XML is far more viable. In order to deploy legacy technologies the natural structure in information has to be dismantled and atomized; there is no such compulsion in XML. In legacy technologies closely related things have to be scattered in multiple destinations, thereby increasing the complexity in bringing them back into local syntactic contexts, making software development complex. In XML one never runs out of "local space", an element can always be expanded within a local context by adding attributes and nested elements. XML is not compulsive about uniformity and allows object instances in the same collection to range from simple as atomic values to as complex as necessary. Information tends to reside in its most natural context and its evolving complexity is absorbed easily and locally without massive displacement and need for a global redesign. Path expressions provide natural handles for values and their collections. (See Appendix A for an interesting example.) The counterpart of path expressions in legacy technologies can have potentially unbounded syntactic complexity.

As stated above, with automation some advantages for the paper-based systems that we took for granted have become distant. This include loss of organizational memory, lack of clarity on routing work for approval, and ensuring that an organization's procedural requirements are met. Furthermore, reliance on email has complicated the determination of the status and needed residual work at any given moment of time. The current state of approvals is often not clear and sits in spaghettis of emails that have to be manually browsed before clarity emerges, and that too for a short period of time. Even a list of participants in a process and their roles are not immediately available or clear.

The description of a course has several components, requiring different people to interact in the development and approval process. The deadlines also vary from one field to another. Sometimes a task has to be repeated multiple times on parts of participants. For example, a department curriculum committee may have made a decision about a course and may like to enter it in the catalog without having to worry about multiple deadlines for various components. One should not have to wait for an opening date for entering the information. It would be best if the next catalog always remains open for editing. This idea prompts us to actually extend the conventional concept of organizational memory: obviously it consists of past memory but it also extends into future until a task is completed. The components that do not get incorporated in the next catalog should automatically carryover to the catalog following the next catalog. In this way, by eliminating start deadlines one opens a system for recording future memory.

Current systems do not facilitate collaborations as smoothly and widely as they could. For example, at department level every faculty may be allowed to participate in modifying courses. A well designed catalog system can eliminate the need for calling a meeting on every small issue or collecting inputs from faculty via email and then having to collate them. The information should always reside in a collated form. XCCat provides a town-square like environment, where all participants can work collaboratively. All information, dynamic reports, and identity of participants are always visible. This is appropriate as catalog development is not a highly secretive process. All parts of the catalog should be visible to all, although editing is privileged to a specific participants.

From the point of view of catalog development it is useful to think in terms of programs rather than departments. Most programs are offered by single departments (hence single colleges). But some programs are jointly offered by multiple departments residing in multiple colleges. Every program designates one faculty member as a liaison in the development of the catalog. This person interfaces the curriculum committees in the program and colleges. The college level curriculum committee interfaces with the Faculty Senate, the body that is overall in charge of catalog development. In addition, there are other bodies that look into specific aspects of the catalog. For example, in case of dual listed courses – the courses that are available to undergrad as well as grad students – the Graduate College ensures that the graduate student side of the course meets some additional requirements. We can divide the participants into three types of participating units: university at the top, and colleges and programs below it.

In this day and age, query for the content of information should be easy, but that seems distant due to two reasons. First, the information may reside on a platform that does not even support query. The second reason is that the organization of information in legacy system may become so horrendously convoluted that there may be little incentive to make it available for on-line query. Thus we may fail to tap the potential advantages of computer and connectivity technologies adequately.

The catalog system is fragmented in ways that is not necessary. For example, experimental courses at Iowa State may form a separate catalog without any compelling reasons. On the contrary, XML encourages us to expand a system to bring more content under a single banner. As the local information is not dismantled to begin with, it simply becomes a part of a global context without a change in semantics. Even syntactic handles to access the information remain largely stable. (See an excellent example in Appendix A.). Indeed we bring multiple year catalogs under XCCat. This will have several benefits. First we note that the degree requirements for students although primarily rooted in the catalog at the time of their admission, adjustments span multiple catalog years. Another advantage is we immediately gain ability to query the catalog evolution spanning multiple years. One could pose queries such as evolution of prerequisites of a specific course or the growth in number of courses offered by a program. A container for queries and their development is included in XCCat. All

participants will be able to pull the reports when needed, even on the fly: e.g., in a meeting. The framework would be useful for departments, programs, colleges, the Graduate College, curriculum committees at various levels. Registrar, Provost, Faculty Senate, and Board of Regents (the ultimate governing body for all three state universities in Iowa).

Current user interfaces are often inadequate and tend to separate factual information from routing and approval requirements and views posted by participants. In this case we feel that the use of legacy systems convolute information so badly that to bring related things together in a single framework becomes a cumbersome task with little promise of potential gains compared to the effort that would have to be put in to reap such gains. In a nutshell in XCCat, due to its roots in XML, all information for all users, at all times remains collated at a single destination, that can be accessed in their most natural ways through customized graphical interfaces to facilitate text entry and editing, approvals, deliberation, query, and publishing in one unified framework without any loss of organizational memory.

The rest of this thesis is organized as follows. In Section 2 we describe the storage organization of the whole catalog system in XML. In Section 3 we discuss implementation, including several user interface. In Section 4 we give a brief description of the project. Section 5 mentions prior and related works. We conclude in Section 6 where we assess XCCat prototype and future work.

CHAPTER 2. STORAGE MODEL

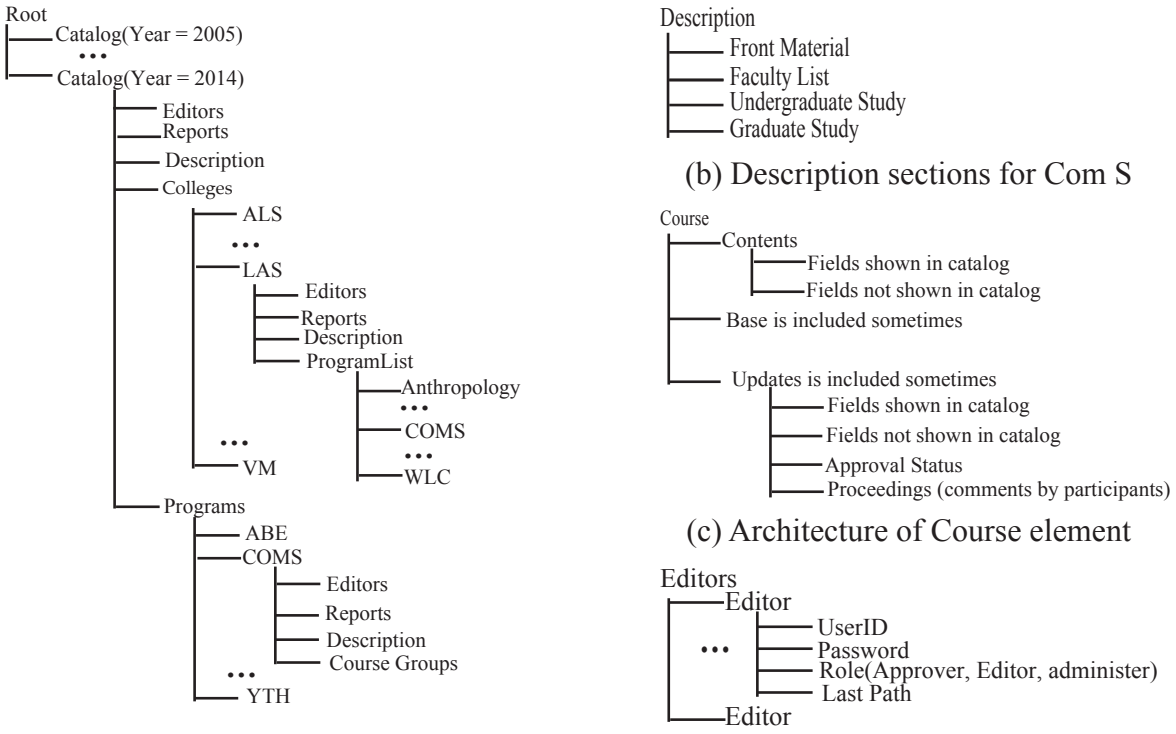
The advantage of XML as a database platform has been explained in introduction. As stated before, this thesis reports XCCat, our prototype for course catalog. XCCat deploys XML for providing an enterprise solution for the management of course catalog system. In this chapter we consider the architecture of the storage used in XCCat.

Figure 1 shows the storage architecture of XCCat as a single XML document. The architecture brings catalogs spanning several years in one place: as an example, the figure shows years 2005-2014. Incidentally, an academic year such as 2011-12 is referred to as 2011 in the catalog.

We recall the hierarchical organization among participant units. At the top of the hierarchy there is the university as a participant unit, and under it are college and program level participant units. The organization of a year directly reflects the hierarchical relationships among the participant units. The university is at the top and colleges and below it are colleges and programs.

Each participant unit consists of Editors, Reports, and Description elements that will be explained shortly. In addition, each also has additional relevant information. Most noteworthy are the Courses under program groups, under which we have individual named Course Groups courses, that in turn consist of course element. The organization of individual courses turns out to be a major task and it will be dealt with in considerable detail.

- The Editors element consists of a list of users that play the role of editor for at a participant unit. In XML it consists of User elements. User elements contain userid, password, and privilege of editors. The list of editors is visible to all catalog participants. Currently, we envision that this list is maintained by a Catalog Administrator at university level, who too is enrolled as a user.
- The Reports element is a container for a set of queries relevant at a participant unit. Currently, these queries are expressed in the XQuery, the W3 recommended query language for XML. It is envisioned that participant unit would store useful queries here. Only



(a) Overview of storage model in XML

(b) Description sections for Com S

(c) Architecture of Course element

(d) Editors

Figure 1. Overview of storage model in XML

authorized user can add, delete, or revise query in the Query element. However, the read, execute, and copy access is allowed to help useful queries to be shared among other participant levels. The result of a query is a report that can be displayed on the fly or saved somewhere in user area.

• The Description element contains the totality of text-based information belonging an participant unit. The number of named sections under the Description element can vary among organizational units. No standardization of formatting is imposed. Perhaps this may evolve in future and then this part can take a more definite shape. The participant units in a university are mutually independent bodies. They can distribute the workload in a way they find it fit. They may designate different individuals or committees, possibly with editing privileges, as in-charge of different sections. For example, the description sections belonging the Com S program, shown in Figure 1(b) are Front Material, Faculty List, Undergraduate Study, and Graduate Study. A very easy to use feather-weight editor, to be described later, is included in XCCat for creating and updating description sections.

• In addition to the above elements, every participant unit has an elements that is specific to its needs. The university unit has Colleges and Programs elements than contain lists of names of colleges and programs, respectively. The college level participant unit has Program List, containing a list of names of programs in which the college has a stake. In Programs participant units the Course Groups element will be described shortly.

2.1. Rendering of text

A proper rendering of text requires a framework such as a text editor. For maintenance of consistency the most important are paragraph styles, character styles, and page styles. These are very poorly understood and highly confused in Microsoft Word [8] and highly well understood in FrameMaker [9] representing two extremes. A reasonable compromise is reached in html technology through the use of Cascading Style Sheets (CSS) to represent paragraph styles. It is neither easy nor desirable for course catalog to use these technologies. The whole catalog uses only a small number of paragraph, character, and page formats. A feather-weight editor with our own XML tags as an integral part of the catalog system is a better solution and included in XCCat. The editor renders ordinary paragraphs in description sections, course groups, and individual courses. Currently, we have not considered character and page styles.

The editor is based entirely on XML elements of the form <p style = “specific style”>para-content</p>. A library of specific style, expressed in CSS like syntax, to be used in the catalog is included. Besides information such as font type, size, face (bold, italic, etc.), vertical spaces before and after the paragraph, and a feature is that expresses “end this paragraph without a line feed”. This helps us realize every field in a course as a paragraph and at the same time render the whole course that has visual appearance of a single paragraph. The styles for various paragraphs in courses are implicitly hardwired, but in the description sections they are chosen from a drop-down list of built-in styles in a library.

Thus our solution for the editor is entirely XML based which is perhaps the best thing to do: even Word and FramaMaker have progressed in this direction. We have not yet included character formats. Which means for example that in a paragraph a specific word cannot be highlighted, for example by italicizing it. The interplay between paragraph and character

formats in Word is confused to such a high degree that in relatively small documents dozens and even hundreds of spurious styles are likely to be generated and bewilder a user. On the other hand XML technology is so well thought out that a rudimentary addition of character styles in XCCat should not be difficult.

2.2. A single description section

A description section is simply a sequence of paragraph elements described above. We have not implemented nested paragraph styles such as this thesis document that has sections and sub sections etc. The appearance of these have to be given by proper choice of a paragraph style and enhancing the content. To begin with this is not an unreasonable solution as the description sections belonging to any individual participant unit is not very large. In course of time more features can be conservatively added to paragraph styles.

2.3. Course structure, updates, and approvals

Figure 1.(c) shows the storage structure of container for a course. Every course has a Contents element, but some also have Base and Updates element. Of these Updates element is very elaborate. The motivation for the structure of a course is rooted in the development process of a course leading to its publication.

The contents element of the course consists of various fields of the course some of which are to be displayed in a print copy of the catalog and others to facilitate some organizational memory relevant to that course. If a courses is stable, i.e. a course for which no changes are to be considered during a catalog year, the Container element suffices. The more interesting case is when changes are considered for a course the Base and Updates elements are included.

During any year Y we have a published version of the catalog that is in effect. In addition we have a development version of the catalog for year Y+1 for which the former one forms the base. The purpose of the Base element is to maintain a copy of the contents of the course in the previous version. The updates element in Y+1 year catalog changes dynamically during development, but the Base element does not. The Base element obviously introduces some redundancy: the Base in year Y+1 is same as Content in year Y. But this redundancy is intentional. It it localizes the entire developmental proceeding which become part of the organizational memory of the course that can even be queried.

At publication time for year Y+1 it can happen that no changes need to be made to the course. In that case Base and Updates elements are removed. This can happen due to several situations: e.g. when no change is ever posted or posted changes were withdrawn and no memory of this is to be kept. Thus Base never goes through changes for catalog year Y+1 except it may sometimes be deleted as a whole.

The purpose of the Updates element is to store all information surrounding the proposed changes. Currently we have envisioned it to consists of Proposed Changes, Approval Status, and Proceedings. Put together, Contents, Base, and Updates elements can be viewed as a central public square for all participant units to help them collaborate in development of the course until the eve of publishing deadline.

Publishing the catalog means publishing the finished catalog for year Y+1 and at the same time also publish the development version for the year Y+2. For year Y+1 the Contents are recomputed by applying the updates. Base and the Updates elements are not changed. Year Y+2 carries the residual pending work that cannot yet be made part of Y+1 catalog. The details are shown in Figure 2.

In the above discussion we have mentioned approved and pending changes. Understanding the difference is facilitated by the concept of an approval vector associated with every field. In this thesis we have proposed the concept of an approval vector having the format (☐ | O O O | ☐). the first ☐ is to record approval by liaison of a program, the next three O O O are for use of a college corresponding to Y | N | D (Y = “yes”, N = “no”, D = “differed”). Only one or none of the three radio button can be filled. (If one is inadvertently selected by clicking a radio button, it can be cleared by clicking on it again. This is a choice we provide that is ordinarily not available with radio buttons.). The last one ☐ is for University’s use.

Here is how ☐ can be used by a program, possible a department. All faculty can be allowed to participate by suggesting changes. Even conflicting changes can be indicated by remarks. This can go through some evolution. Finally, the program liaison can edit the proposed changes to bring the final version to an unambiguous state and indicate the end of deliberation at the program level by checking ☐. Only when this box is checked the colleges administering the program should consider the proposal. Such consultation within the department can be

Catalog for year Y
Com S NNN (the root)
  Contents: Various fields of the course (modeled as attribute and element children)

(a) The structure of a stable course

Catalog for year Y+1 during development
Com S NNN
  Contents := Y.Contents
  Base := Y.Contents
  Updates: Changes requested in various fields, approval status, and comments

(b) The structure of a course during development

Catalog for year Y+1 when Published
Com S NNN
  Contents: Y.Contents + approved changes
  Base := Y.Contents
  Updates := := Y+1.Development

Catalog for year Y+2 opened for development
Com S NNN
  Contents := Y+1.Published.Contents
  Base := Y+1.Published.Contents
  Updates := Y+1.Non-approved updates
  If Updates becomes empty then remove the Base and Updates elements

(c) Structure of a course at publication of the catalog

Figure 2. Course organization during development and after publication

done without having to meet in person and without having to use emails. Similarly the college can deliberate by posting comments and finally enter Y | N | D. The university can also post comments and eventually select □ or choose not to. Note that the approval vectors for different fields are independent. Some fields may require additional components. In the current implementation Dual listing requires approval form the Graduate College.

It is the approval vector that allows us to partition proposed changes into once that can be incorporated into the year Y+1 catalog and those that have to be differenced to year Y+2 catalog. Note that the approval vector may evolve in future. However, the algorithm to split changes among the two catalog years will remain simple relative to the interpretation of the approval vector.

This is where the advantage offered by XML should become clear. In XML, a local context can always be expanded to accommodate something deemed useful and shrunk by releasing what becomes extraneous. One also comes close to human perception that when something is empty it is simply not there and in that case it does not even make sense to ask certain questions. Even a concept goes in and out of existence. The changes in structure can be localized without having to go to square one of in the design process. The storage structure varies from one course to another and still make them look as homogeneous as possible. The algorithms to accomplish organizational tasks would be simple having the minimum local context that can be easily expressed syntactically.

CHAPTER 3. USER INTERFACES AND IMPLEMENTATION

In last chapter, XML-based storage model for XCCat prototype has been introduced. A common GUI is available for the use of all participants. The GUI consists of two vertical panes. The XML elements in the storage are shown in a tree format in the left hand pane in a format that is intuitive to a user. Internally the physical element may differ from how it is displayed for user.

JTree is used to create a customized display of the contents of the XML-based catalog document in a tree format in the left pane. The labels for nodes, their designation as leaf or

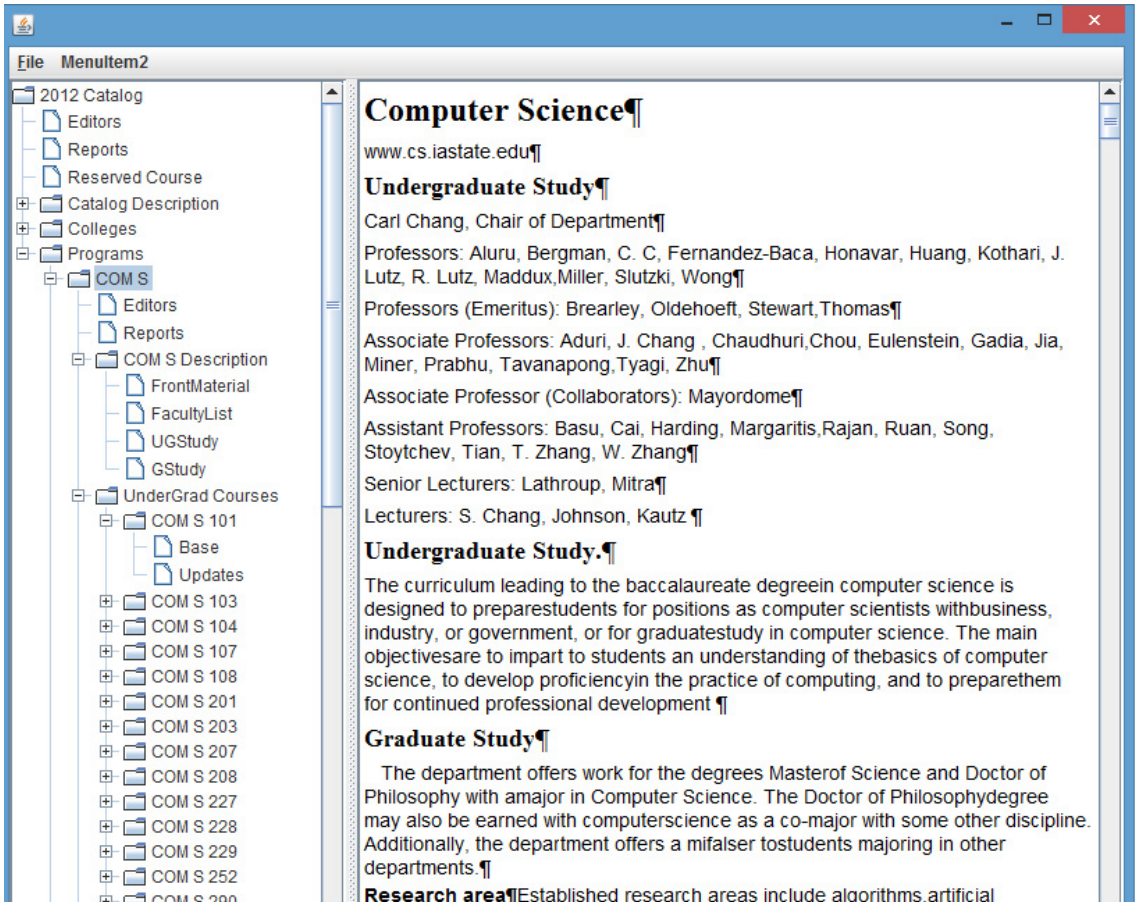


Figure 3. When “Com S” node is clicked by the user

non-leaf, and their inclusion non-inclusion in the tree are customized. JTree adds + / - before the labels of non-leaf nodes to allow a user to expand / collapse them. What is shown in the right hand pane when a node is clicked can also be customized.

A participant can click on any node at year level or below in order to view its contents. As an example the result of clicking on Com S, a program is shown in Figure 3. Thus the entire catalog is visible to all participants. On clicking a node what is displayed varies considerably from node to node and tabulated in Figure 4. It is seen that in every case the most logical interpretation is displayed. A user having editing privileges can continue and edit without going through any extra steps. There are five such cases allowing editing. The case of course group is simple and it is explained completely in the Notes column in the table. The remaining four are covered in detail in separate sections below.

3.1. Editing a specific description item

As we mentioned before we could have simply allowed a single item called *Description*. But often different people are in-charge of different "parts" of it that we have termed *description items* or simply *items*. For example, in a program description, the list of faculty can be considered an item. (At Iowa State this has traditionally been provided by the Provost's Office.) Other examples of in computer science program are ones that are titled "Undergraduate Study" and "Graduate Study" that happen to be under the jurisdiction of undergraduate and graduate curriculum committees, respectively. Thus the ability to divide Description into items and naming them appropriately would help a program manage it better. The choices of items will vary among programs, colleges, and university. Internally an item is simply a sequence of content . The WYSIWYG editor will only display the contents and use the associated paragraph styles. The styles are represented CSS style. If one is satisfied the item can be saved by pressing [Save] button to save the edits.

3.2. Editors elements

Every participant unit: university, a college, and a program has an Editors element. Any participant can click on an Editors element to see its contents. The contents are formatted as a form as shown in Figure 5. In the current version, all Editor elements can only be edited by

Clicked element	What is displayed	Notes
Catalog Year	The print version of the entire catalog	Cannot be edited. Possibly some performance issues.
Colleges	Combined print version of all colleges	XML can be edited manually. Need to update the list of colleges should be rare.
A specific college	Print version of information about college	The element cannot be edited at this level. List of programs can be updated manually.
Programs	The entire print version of all programs including descriptions and courses.	(a) There is possibility of some performance issue in displaying entire contents. (b) XML can be manually edited; the need to update a list of program names would be in-frequent.
A specific program	The print version of the program including description and courses.	Cannot be edited at this level.
Editors	A custom-designed GUI panel contains the list of editors and privileges.	Currently only Catalog Administrator can enroll editors. See Section X.Y.
Reports	A custom-designed GUI panel contains a list of all queries, including their informal description and XQuery code.	All can browse, execute, copy queries, and save reports. Only the participant unit can update. See Section X.Y.
Description	The combined print version of all description items	Cannot be edited at this level.
A specific description item	The print version of the item	Someone internally assigned by a participant unit to the item. See Section X.Y.
Course group	Print version of the title of the group followed by a list of courses	The title can be edited and a number for a new course can be created by starting with a right click. Courses in the list cannot be edited at this level.
A specific stable Course	Print version of the course	By definition it will not be edited.
Root of a specific course being updated	Print version of the course	Automatically changes only at the time of publishing the catalog.
Base element of a specific course	Print version of the Base course	Remains frozen for years Y+1.
Updates element of a specific course	A formatted form.	This changes with deliberations about a course proceed and visible to all. Collaboratively edited by many parties. See Section X.Y.

Figure 4. What happens when elements in XML tree are clicked upon

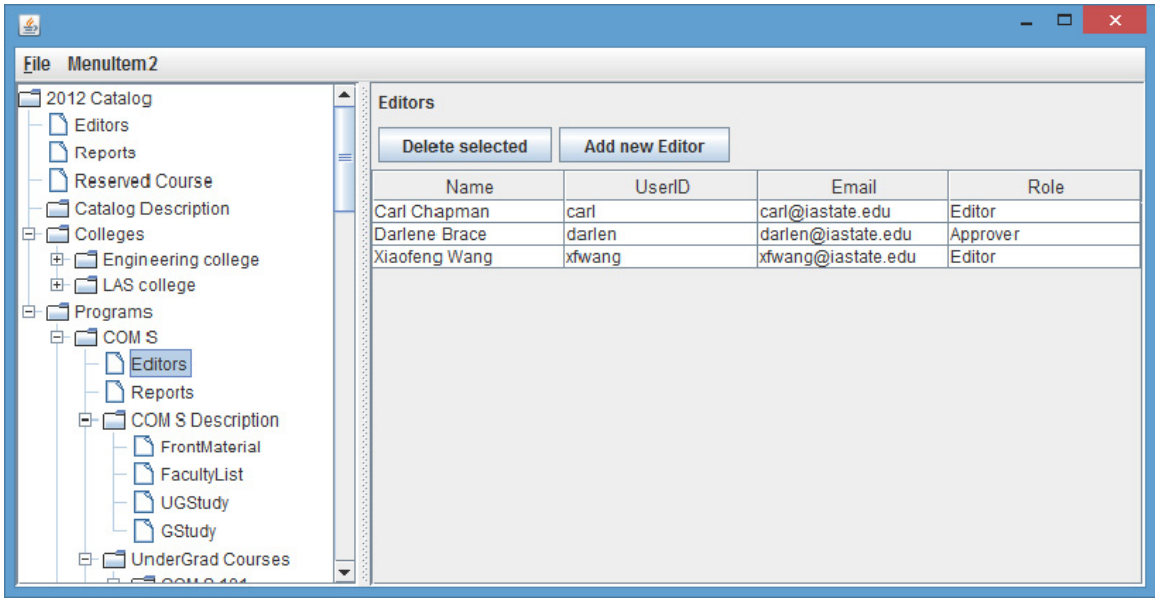


Figure 5. Panel for List of Editors

one university level central authority, called Catalog Administrator, or simply Administrator. The Administrator can edit any Editors element to manage enrollments of participants and access control.

A participant has a name, a login-id, and role. The roles are Editor, Liaison, and Administrator. A program may have several editors, but only one liaison. Colleges also have several editor and approvers to share their work. The Administrator also enrolls herself as an Administrator. Thus it is always clear who to contact to enroll editors and liaisons at lower levels. Some functionality can be added in future at program and college level Editors elements to make it easy to initiate such requests.

3.3. Reports elements

Reports element is available at every participant level. Anyone can click on any Reports element in the XML tree to see its contents. The display consists of a panel that allows interesting queries to be maintained, and executed. (See Figure 6.) The queries can be seen by all participants at all levels. They can also copy queries (one at a time) and paste them in their own Reports element that they have editing privileges for. They can also execute existing queries and develop new ones. The queries require and XQuery engine that may need

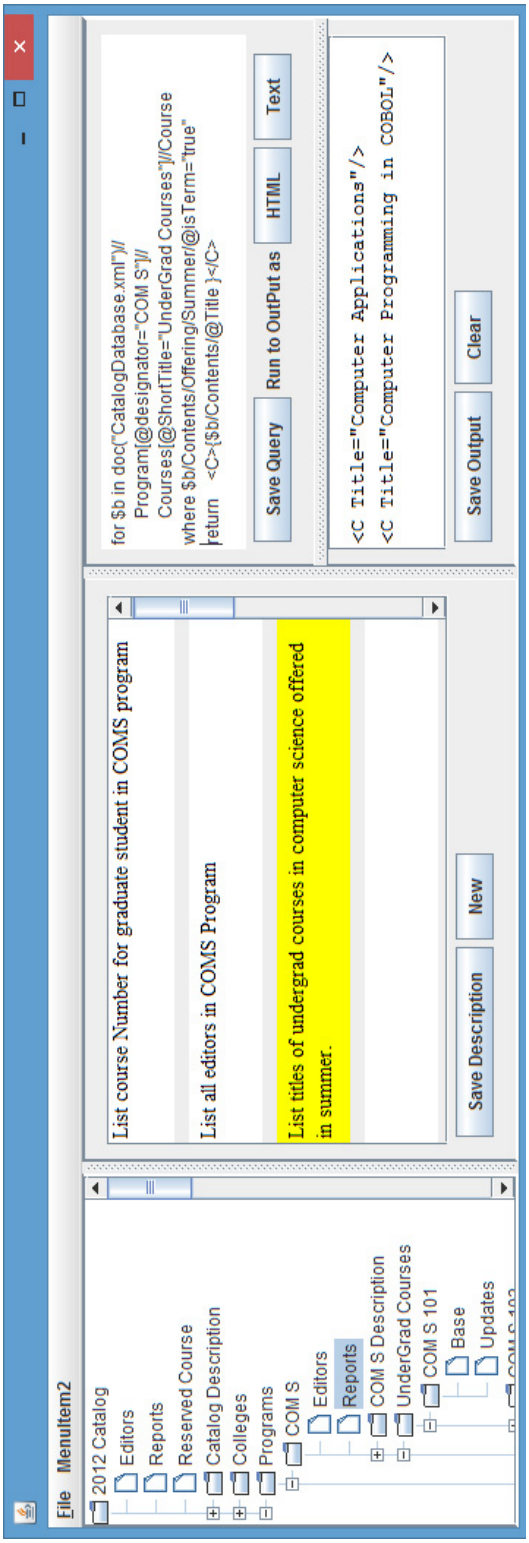


Figure 6. Panel for maintenance of reports

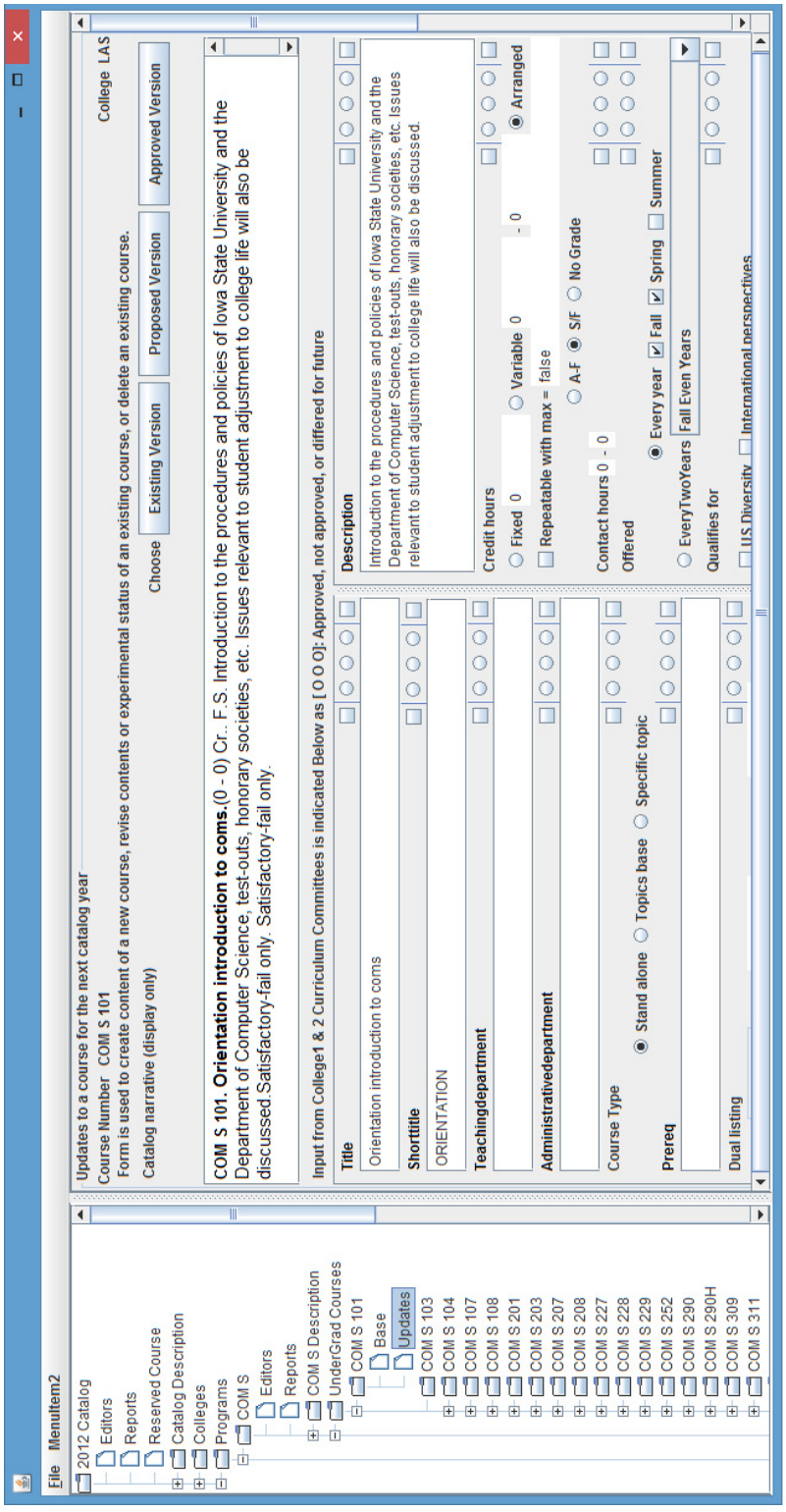
university wide license. When a query is executed the output can be considered a report. A report can be stored as a file. All this functionality is included in the Reports panel. Here are some examples of queries.

- List all the name of all programs under the LAS.
- List number of courses offered by Com S in 2015 Spring Semester.
- How many credit hours in 2015 Spring Semester.
- What courses need to be scheduled for next Fall in computer science?
- How many undergraduate courses are offered by each program?
- What new courses were introduced last year?
- How many credit hours have been added (or reduced) in the whole catalog?
- What is the approval status of all courses?
- List the history of prerequisites for Com S 311 in last 5 years.
- List names, emails, roles, and affiliation of all editors.

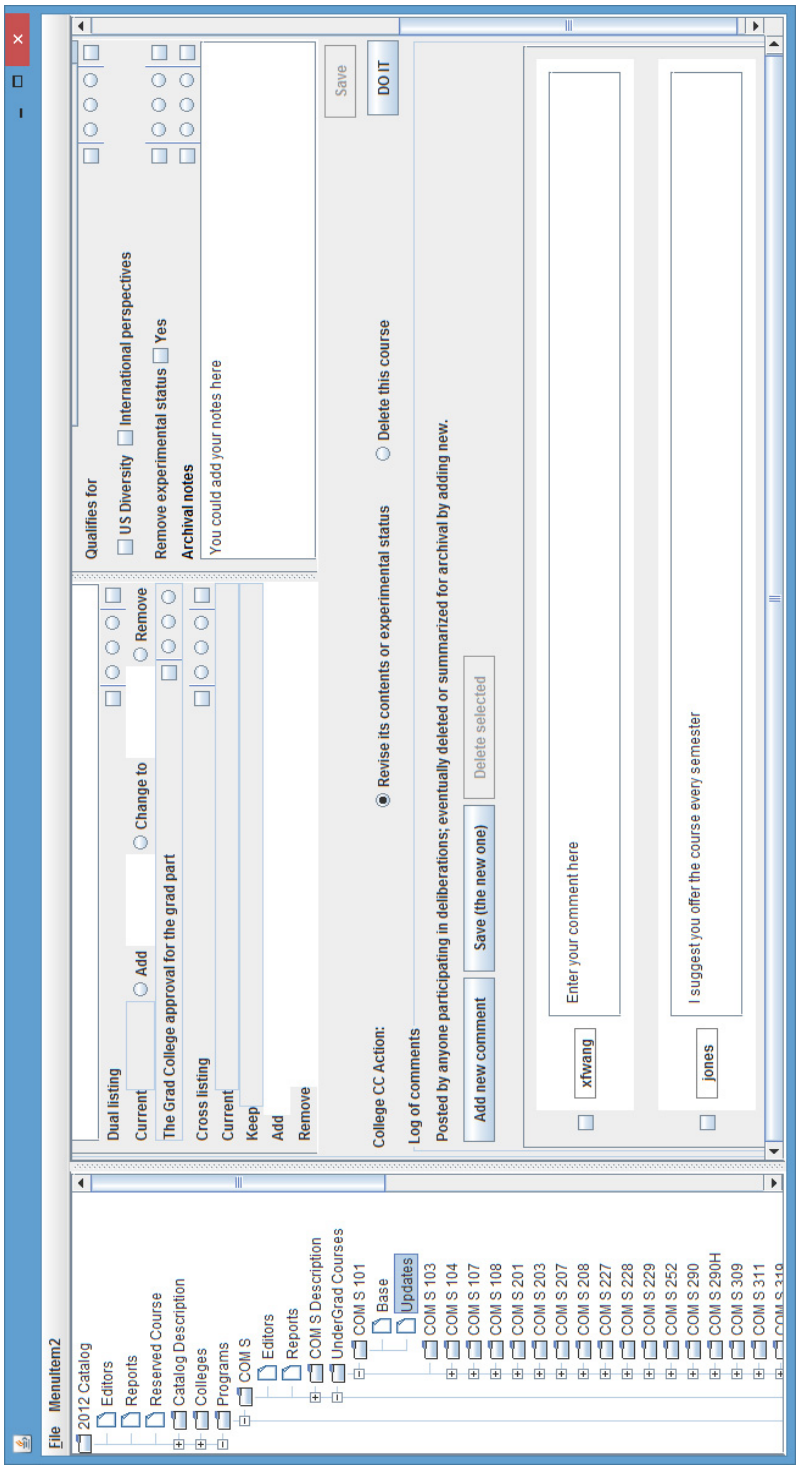
Note that some queries take advantage of the fact that the catalog system in XCCat covers several years. XQuery is a very powerful query language compared to which SQL seems rather primitive and simplistic. After some learning curve, XQuery queries, due to natural structure of XML documents, are actually easier to write than SQL. All participants do not have to develop expertise in XQuery or XML. The queries can be developed by technical staff. The XCCat prototype makes it easy to share queries among participant units. A query developed by one program may be interesting to other programs and may require minimal changes. An XQuery query can be used to create reports that are internally formatted as text files or html documents. If necessary other tools, such as XSLT, or DOM can also be made available from the Reports element in future.

3.4. Course elements

Clicking on a course number (e.g., Com S 311) will simply display the print version of the course. If a course is stable, i.e., if it is not undergoing a change in the catalog cycle, there is no further information in the course element. Internally it is a complex element that may contain other fields that are not displayed. An example is Teaching Department for the course. For example some courses in software engineering are taught by Computer Science



Upper part of Figure 7



Lower part of Figure 7

Figure 7. Panel for enterprise-wide deliberations on a course

Departments, some by EE CprE, and some by other departments. In addition, we have also included a field called “Archival Notes” that helps in maintaining organizational memory related to a course. The courses that are changed in a catalog cycle include significant opportunities for retaining organizational memory. This mechanisms can be further extended in future.

It is the Updates element that is the most interesting part of the XCCat prototype. Clicking on it opens a panel shown in Figure 7. Its consists of three parts: contents, Approval vector, and Proceedings. Contents consists of fields as in any course element. With every field an approval vector is included. Some fields mat have additional approval components. The Proceedings consists of comments. The contents are edited by a program, The parts of approval vector are shared by program, college, and university having authorization for specific parts of the Authorization vector. The comments can be posted by any participant and does not require even program level authorization.

The panel creates a town-square like environment where different participants can come together. Comments can be added by a college stating why a certain field does not meet an approval. The program can then change the field to renew its request for approval. The system can be extended further to make this process more user friendly. The whole proceedings can be saved. But this can sometimes become overwhelming and actually become a distraction. Therefore we allow comments to be deleted. Sometimes multiple comments can be collapsed together. In the end, before the eve of publishing, one can go through a cu-ration process and only retain the comments that help maintain organizational memory.

When the catalog is published, a course element is recomputed. The storage architecture makes these computations completely local. At the end of the yearly cycle the finalized catalog for year Y+1 and development version of catalog for year Y+2 are published simultaneously.

3.5. User interactions in the course panel

As a user interacts with the panel for a specific course the XML representation needs to be updated. The fields are divided into several types. The procedure for performing the updates

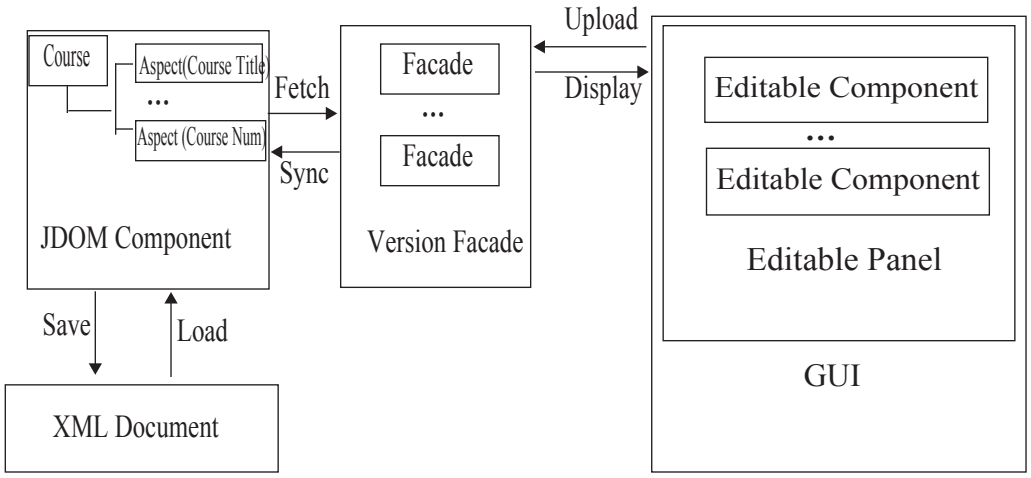


Figure 8. Diagram for all component

vary from type of field to another. A good choice for pairing GUI fields and XML elements is an interface called Facade. Different implementation of facade facilitate handling of different types of GUI fields. For example, whereas updating title and prerequisites fields require a certain instance of a facade, updating cross listing and offering fields require a different instance of a facade.

In order to process the XML document containing the catalog it is first loaded in main memory where it is represented as a JDOM object.

When editing request of a course is invoked, the corresponding element such as a course or a program is identified by internal XPath query on Jdom document. Then a Version Facade, which contains a complete set of facades for all atomic aspects of this course or program, is created by pulling all information needed under the element in Jdoom dcument. In the subsequent step, all Editable Components corresponding to the facades are created on and shown in the editable panel of GUI. The Figure 8 shows a typical architecture of data model for Editable panel of course information. An item, such as “course title” or “curse number” is an atomic and editable aspect for a course. It has a corresponding Facade in Version Facade and an Editable Component in Editable Panel. Any change that made by editor or approver is listened to locally by Editor Component and uploaded to the corresponding Facade. At same time, Facade itself will check whether it has it has the same information as aspect in JDom document. If not, a save button on Editable Panel will be enabled and synchronization can be

made by clicking on the button. The synchronization process concludes an override action of the information in corresponding element of JDom document, and a save action from JDom document to XML file on disk.

CHAPTER 4. DISCUSSION

No high level tools can solve a problem that cannot be solved in assembly language of any computer. The high level tools help in simplifying the design, development, implementation, and ongoing maintenance of applications. No one platform is suited for all application; so for a given application appropriate choice of a platform is important. This thesis demonstrates that XML is a hand-in-glove match for a course catalog system.

Normally a catalog is confined to information about curriculum and courses for a given year. In our XML-based representation, the organization of a yearly catalog mimics the organizational hierarchy consisting of university at the top and colleges and programs below it. A print version of the catalog becomes a very simple exercise and this is not our major preoccupation. We want not only meet but exceed the features and advantages offered by paper-based technologies and try to harness the full potential of computer and connectivity technologies. Use of XML makes a yearly catalog a live database that can be queried in powerful ways. Yet, this is just the starting point. We extend this in several orthogonal ways to provide an enterprise solution for bringing the whole catalog process under an single umbrella. We bring proposals for new courses – that may or may not be experimental – seamlessly under the catalog.

In order to include the information about changes in course content under a yearly catalog, we add two elements to a course – one containing a local copy of the old version of the course and the other containing all proposed changes. At the time of publishing the new version of the course is computed locally by applying changes to the old content. The two child elements are not removed. This also supports powerful query of all changes in courses made during a catalog year without having to look elsewhere. Localization of context is great benefit offered by XML. The organizational memory of changes to courses becomes clear without having to resort to the previous year catalog. The query also becomes easier and eliminates need for a join operation.

For every field in a course that is being changed, we also add a vector showing the status of approvals. This allows every field to be treated independently as the deadlines and the persons

who scrutinize changes can vary from one field to another. The GUI shows the current status of a course. It also allows what if scenarios implemented as buttons for on the fly visualization of current version, the proposed version, and the approved version without having to publish the course.

A list of participants is included at every organizational unit. The enrollment of editors is currently viewed to come under the university level catalog administrator. All participants are visible to everyone but the enrollment is authorized by the Administrator.

We also add a discussion board for every course that is undergoing a change. Anyone can post observations, suggestions, and reasons for the lack of approval in form of comments. This creates an atmosphere that is akin to a public square. The square remains alive during the entire development process. This almost eliminates the need for emails, that would simply be a horrible way of conducting such business. The square encapsulates all the organizational memory about the proceedings in one central place for every course. It always remains in a collated and complete form. There is no need to browse spaghettis of emails to find the current status of the approval process.

Capability is also added for storing useful queries and development of new ones. This is available at all organizational units: university, colleges, and programs. XQuery is a powerful language and this feature would allow a report to be prepared at the click of a button. Everyone has read, execute, and copy access to queries belonging to others. This allows any organizational unit to search for queries that could be useful to them and copy and easily adapt the queries to their environment.

In summary, an yearly catalog includes information about curriculum, courses, people participating in the catalog development process, the proceedings of the course development, a record of all changes, reporting capability for query of contents of the catalog and the changes. No separate handling of new course proposals and experimental courses is required. The information remains in a collated form, eliminating the need for emails and in some cases even physical meetings to conduct the catalog related business. Minimum information is entered and what is not absorbed in the next catalog carries over for consideration for to the catalog for subsequent years. Not only the timetable for entering information is simplified, but

once entered, the system does not forget it. Information remains in a collated form and greatly simplifies the approval process.

Having summarized the concept of a yearly catalog in XCCat we observe that conceptually[1] we have extended the concept of a catalog to include catalogs of past years, the current year, and the development version of the catalog for the following year. This is beneficial as a student overlaps several yearly catalogs. In XML such multi-year integration is a simple exercise. This also helps us monitor and report catalog information beyond a single year. For example what is the year by year rate of increase in credit hours in each program at the university.

1. All aspects of the catalog have not been implemented but a substantial core has been completed. Furthermore, the path to complete implementation is fairly clear.

CHAPTER 5. PRIOR AND RELATED WORK

The idea that XML could be used to turn a course catalog into a live query-able database was first pursued in Com S 562, a course in Database Implementation at Iowa State university with Rahul Ravindrulu several years ago [6].

Serious implementation was started by Carl Chapman, an undergraduate student in Software Engineering, during the 2012-13 academic year [7]. He initiated the GUI design with the left pane showing the catalog as a tree. In most cases. on clicking a node in the tree would display the contents of the node in the right panel. The right panel for updating of course field-by-field via a form was initiated by him as well. This form also allowed users to post comments. The framework for executing the queries was also initiated by him.

Under this thesis we have expanded the system developed by Chapman. The structure of the course was expanded to include the Base and Update elements to facilitate organizational memory of changes. The field level concept of authorization was also added to courses. Concept of publishing has been developed here. The course level publishing is partially implemented here but the catalog level published needs to be carefully expanded upon and implemented.

Redesign of fields in course description was undertaken with the help of the catalog administrator at ISU. The graphical representation of these for updating required considerable thought. The original data was in a spreadsheet which was first cleaned up a bit, then converted to a spreadsheet in XML format, and finally a clean loading in to an XML document was undertaken. It is interesting to note that the loading was done by a query in XQuery language without direct use of DOM API. This is quite remarkable as XQuery is a very high-level language. (High level means more user friendly but not more powerful).

Ability to view existing, proposed, and currently approved versions was added to Course Development Panel. This required computations relative to Base, Updates, and state of approvals to be implemented. A simple algorithm was obtained due to locality of storage afforded by XML. The buttons help in course development process.

For text and course description editing and rendering a feather weight embedded editor was developed that is very easy to use. Concept of paragraph formats was developed. A library of paragraph formats in CSS-like syntax was developed. An interesting component of a paragraph format is one that suppress the line feed after the paragraph. Using this feature the course description gives appearance of a single paragraph, which internally consists of several paragraphs, one for each field of the course. This allows rendering of text passages and courses to be treated uniformly.

It is proposed that the catalog should span several years. When publishing a catalog, the development version of a future catalog is supposed to be published immediately where pending work is incorporated so the catalog development process never needs to be interrupted.

This work seems to have no lineage other than the zero information loss model [10]. This has been explained in Section [6].

CHAPTER 6. CONCLUSION AND FUTURE WORK

We feel that XCCat offers a viable road-map, a fair part of which has been implemented. In this section we highlight some features and a enumerate of directions in which it can be continued.

It is clear that all users of the catalog see and interact with the parts they need to and also see the big picture where their role fits. The big picture in front of a user consists of two parts that can be termed vertical and horizontal. The vertical part is the hierarchy where a user fits, and in an unexpended view this is part is rather small. The horizontal part for a college and program are other colleges and programs. Their visualization can suppressed without disturbing the big picture qualitatively. This needs some thought, development, and implementation. The print view that is obtained when one clicks on the root of a given year is normally what we think of the public view. Thus public can be defined as a user who only sees the root of the XML tree. Different users see different parts of the tree. A similar concept of a user has been introduced in [10].

There is only one “goto” place for all users of the catalog including the public. To address security and privacy issues, parts of the catalog can be mirrored at different places without conceptually changing the one goto place. The one goto place includes what is conventional as well as new course proposals and experimental courses. Moreover, catalogs spanning multiple years are brought under the same root – that is the go to place. This destination remains stable and does not change with time either.

Additions, deletions, and updates to courses have been modeled in as uniform manner as possible. This should help in maintenance of XCCat. The concept of publication has been developed carefully and the development version of a future catalog is supposed to be published simultaneously when publishing the next catalog. The catalog development can be thought of a linear ongoing non-stop process where changes that meet approval are simply removed once a year.

The system has memory of past, present, and future (consisting of pending changes). One can query for the not only the information but also changes and who made those changes to

the information. By executing queries, reports can be generated on the fly for the organization memory. Even development of queries becomes an ongoing process. The queries can be shared among users for immediate use, customization, or further development. The concept of context can be encapsulated in queries to make them independent of programs and colleges.

The concept of approval vectors greatly eliminate the need for tracking in rigid ways. A lower user in the hierarchy simply deposits information any time he/she finds convenient and the upper users simply pull the desired information for proceeding with approvals.

The user interface is intuitive. One simply clicks on nodes in a tree to browse and edit. Every node has the most logical behavior possible to cater to user needs. Currently even to find who are in-charge of which portions is not easy, there are multiple goto places that are not stable. Currently all editors are enrolled by university level Administrator. In future some of this control could be released to program and colleges to provide them more autonomy and internal flexibility and encourage greater collaboration.

A feather-weight editor has been included. This can be used directly to create description items and render them. It is also invoked internally to render print version of courses. A catalog of paragraph styles in CSS-like syntax is maintained and can easily be extended to include any new paragraph style. Our light-weight editor does not include character styles. Even a rudimentary support for character styles would be a good enhancement. It would be good to add a spelling checker.

How to host XCCat has not been addressed. We imagine a centralized location where it can be housed. We have not looked deeply into performance issues. JDOM requires an XML document to be loaded in main memory. We do not see this as a serious obstacle as size of the catalog document is rather small. We expect the entire catalog for one year to be about 20 MB. Even 10 year coverage would span only 200 MB. Nevertheless this issue must be examined carefully.

Some embedded technologies, e.g. an XQuery engine would need licensing. Although we have a starting point for the approval process, but this would require a more detailed scrutiny. However, the use of XML ensures that any changes or addition can be absorbed easily. Recovery and concurrency should also be studied and implemented. We do not expect the

concurrency (parallel use by multiple users) to be problematic. Locking at course level would perhaps yield a good solution. We have allowed the description belonging to the university, colleges, and programs to be broken into smaller items that can internally be assigned to specific individuals. These items can also be considered granules for locking.

Use of XML has simplified the structure of a course quite significantly when compared to its spreadsheet based representation. For example, cross listing of a course has been modeled as a simple element allowing one or more courses to be cross listed. In the spreadsheet it consisted of 10 columns to allow up to 10 courses to be cross-listed. In XML all the cross-listed courses are reached via a simple variable, something that becomes far more complicated in a spreadsheet. Many such examples can be given.

Once a good Catalog project is successful, other systems can also be considered for XML-ization. This would make sharing of information easier and perhaps more automatic. For example schedule of classes to be offered every term, can benefit from it.

Many other features can be added. For example, the deadlines for changes in a course vary from one field to another. A button can be added to display the deadlines in their physical context where the fields appear. An XML Schema should be developed to ensure that the catalog meets the organizational requirement. For example this can ensure that all cross-listed and dual-listed courses actually do appear in the catalog in appropriate places and form. What cannot be captured by XML schema has to be validated manually. It would be good to add the concept of mailbox for users where they can see pending work and deal with it in a stream. It would be good to add tabbed displays when appropriate. On the other hand users' need for tabs is greatly eliminated as these are seen in the XML tree on the left.

Once a catalog system is in place its scope can be expanded. For example accreditation of programs is a major preoccupation at universities. The catalog can perhaps be queried in the XQuery query language to verify if a program is meeting the accreditation requirements. The understanding of the requirements would need to be xml-ized for this to work properly. It seem that at the time of publication of a catalog such auditing can be done at university level for all programs. Another example is what is called Degree Audit at ISU. This is a procedure that allows a professional advisor to make sure that a student is progressing normally and will

complete degree requirements in time. This would require information about the courses taken by the students to be understood more clearly. Another example is the schedule of classes.

The results obtained in the catalog project are similar to those in [10] where the concept of zero information loss was introduced in temporal databases. There a model and a query language framework was provided where data is never lost, and one could query data, database history of data, query updates and circumstantial information them, and query past queries. The term zero information-loss was used to capture the contents of a text-based transaction logs by converting them into databases that could be queried. The zero information referred to the fact that the whole transaction log could be restored from the database character-by-character and hence there is no loss of information relative to the transaction log. This work also introduced the concept of a user based upon what the amount of information that could be seen by the user. The zero information loss is akin to what is now known as provenance in databases.

APPENDIX A. WHY XML IS EASY TO USE

XML brings machines and humans closer through rich vocabulary that is equally well understood by both. For example in an XML-based library, the *list titles of books authored by Neal Stephenson* may be captured by the XPath expression //book[author =“Neal Stephenson”]/title. (See [1]). This can be considered analogous to an expression such as (– b + sqrt(b^2 - 4ac)) / 2a. When compared to the XPath expression, an SQL-based solution will be enormously more complex. If one were to express in Java, plausible code in JDOM is shown in Figure 9. One may view the complexity signature of the Java code as {{{{}}{}}{{}}}, where {} represents a loop or a conditional. This is far more complex than linear code {}{}{}{}{}{}{}{} and nested code {{{{{{{}}}}}}}. Furthermore, the XPath code would remain robust under some changes in structure, e.g., if the library were to be replaced by a consortium of libraries.

```
ArrayList result = new ArrayList();
NodeList books = doc.getElementsByTagName("book");
for (int i = 0; i < books.getLength(); i++) {
Element book = (Element) books.item(i);
NodeList authors = book.getElementsByTagName("author");
boolean stephenson = false;
for (int j = 0; j < authors.getLength(); j++) {
  Element author = (Element) authors.item(j);
     NodeList children = author.getChildNodes();
     StringBuffer sb = new StringBuffer();
        for (int k = 0; k < children.getLength(); k++) {
              Node child = children.item(k);
              // really should to do this recursively
              if (child.getNodeType() == Node.TEXT_NODE) {
                    sb.append(child.getNodeValue());
               }
         }
        if (sb.toString().equals("Neal Stephenson")) {
                    stephenson = true;
                    break;
         }
   }
   if (stephenson) {
      NodeList titles = book.getElementsByTagName("title");
       for (int j = 0; j < titles.getLength(); j++) {
       result.add(titles.item(j));
       }
   }
}
```

Figure 9. JDOM Java code for a simple XPath expression

BIBLIOGRAPHY

[1] www.ibm.com/developerworks/library/x-javaxpathapi/index.html

[2] XML Path Language (XPath) 3.0. http://www.w3.org/TR/2014/REC-xpath-30-20140408/

[3] XQuery 3.0: An XML Query Language. www.w3.org/TR/2014/REC-xquery-30-20140408

[4] Easy Java/XML integration with JDOM. http://www.javaworld.com/article/2076294/java-se/easy-java-xml-integration-with-jdom--part-1.html

[5] JTree - Oracle Documentation. http://docs.oracle.com/javase/7/docs/api/javax/swing/JTree.html

[6] Rahul Ravindrulu, a student in Database Implementation course who the catalog project was initiated.

[7] In Fall 2012, Carl Chapman implemented a part of what is now XCCat.

[8] Microsoft Word. www.microsoft.com.

[9] FrameMaker. www.adobe.com.

[10] Gautam Bhargava and Shashi K. Gadia. Relational databases with zero information-loss. IEEE Transactions on Data and Knowledge Engineering, Vol 5, 1993, pp 76-87.

ACKNOWLEDGMENTS

Firstly, I would like to warmly acknowledge Char Hulsebus at Iowa State University. She provided us 2014 catalog information that was critical for our project. She also explained the meaning and intricacies of several terms in the catalog. Her help was indispensable for the project.

Secondly, I would like to express my deepest appreciation to my advisor, Dr. Shashi K Gadia, for his guidance, supervision, and persistent help. His enthusiasm for xml and databases kept me constantly engaged with my research, and his personal generosity helped make my time at Ames enjoyable. His encouragement and guidance was particularly helpful when I was new to this field. Also special thanks to his comments on the thesis.

In addition, my thanks go to all members in database group. Xinyuan from who I learned good coding. I also thank Carl Chapman who set up a basic model for this project that he also implemented. Some functions implemented by him have been very helpful.

Furthermore, I take this opportunity to thank all my family members, for their trust, endless support and understanding.