Architecture evolution of an application: Center in a Box

Mano **Marks**

Chitra **Madhwacharyula**

Christine **Jones**

Robert **Glushko**

Abstract

This paper presents a case study in the evolution of a content management system from a static web-publishing model to a Service Oriented Architecture (SOA). We will describe the development of Center in a Box (CIB), an application created at U.C. Berkeley's Center for Document Engineering to support the semi-automatic creation of customized websites for university research centers. In particular, we discuss our use of the document engineering methodology and our decision to use smallx [smallx] pipelines instead of Cocoon [Cocoon]. and the migration to a modified Atom [Atom] format for storing our content.

Table of Contents

1. Overview

This paper presents a case study in the evolution of a content management system from a static web-publishing model to a Service Oriented Architecture (SOA). We describe the development of Center in a Box (CIB), a lightweight publishing system designed for supporting the semi-automatic creation of customized websites for university research centers. CIB version 1.0 incorporates XML schemas, XSLT, and Cocoon as its core technologies for structuring, processing and transforming web content. CIB 1.0 has a more modular, dynamic architecture than a static web page publishing model.

CIB 1.0 captures the conceptual model shared by small university research centers. The data model is expressed as a set of XML schemas and contains components such as information about the center, people associated with the center, events, news, and resources. XSLT transformations are used to create XHTML, RSS and PDF when requested. CIB 1.0 uses Apache Cocoon that incorporates XML pipelines (hereafter referred to as pipelines) to synchronize each page with a table of contents and an index. This allows regeneration of a website each time it is updated or when new pages are created.

In the spring of 2005, we remodeled the data architecture of CIB version 1.0 using a formal and rigorous Document Engineering approach with the goal of creating a more flexible content model. In CIB version 2.0, we moved from a hierarchical data model to a graph model that emphasizes bi-directional links between object components. We also moved the pipeline to a central role, which allows the user to choose the relationship between two content components by manipulating the order and means of component linkage. Finally, we used substitution groups within the XML schemas so that the model is more flexible. For example, this model allows for the replacement of generic events with more specific events, or a generic person to be replaced with a sponsor, student, faculty member, etc.

With the data model built on a more flexible framework of XML schemas, a significant problem remained. Cocoon's pipelines mixed the assembling, aggregating and transformation of files with the presentation processing. We realized that by separating the presentation processing from the aggregation and transformation processing, we could offer a content service that would provide different views of the information. Specifically, we present information as XML formalized by a new set of XML schemas that can be used by other technologies such as mobile devices, RSS readers, standard browsers, etc. After exploring ATOM as a news vocabulary, it became apparent that CIB‘s content could be envisioned as a series of ATOM news feeds. By using the ATOM vocabulary and its HTTP-based API, the website could be transformed into a set of content services with ATOM wrappers and CIB-typed content. This solution was incorporated into our application since it offered a successful combination of industry standards and specific-use XML schemas.

2. Background

The Center In a Box Project was conceived at the Center of Document Engineering (CDE) at University of California, Berkeley. In the summer of 2003, researchers at CDE were looking for an easy way to post information online about the Center and its projects. After analyzing the peer websites on campus, they recognized that there were over 100 other “Centers,” or small organizations at Berkeley. Each of these Centers had its own web site.

The majority of these web sites had sections for news, people, publications, events, projects etc. However, despite these content and structural similarities, there was little commonality in appearance, architecture, or implementation. Without a campus-wide infrastructure or a mandate to define what a Center’s web site should contain and how it should look and behave, most organizations built their own sites from scratch on a flimsy set of Dream Weaver or FrontPage templates.

Compounding this problem was the wide variance in skill level among the staff and hired students who maintained center web sites. As a result, each center’s site had its own look and feel. Minimal common navigation and user interface metaphors existed amongst the websites and information sharing across the Centers was virtually nonexistent. The CDE researchers sought to apply a document engineering approach that would not only bring rigor to web publishing, but also provide a flexible and reusable framework upon which each of these organizations could build and maintain its site.

The CIB Project identifies and encodes the common conceptual model of a small academic research unit. CIB Version 1.0 consists of a set of XML schemas that define the content model, XML documents to store the actual content and XSLT transformations that automatically build the research center web sites by extracting the relevant content from the XML documents. The transforms create valid XHTML and other formats like RSS and PDF, generating appropriate links and user interface components like tables of contents, links and navigation aids. CIB 1.0 was implemented using the Apache Foundation’s open-source Cocoon project that uses pipelines to synchronize each page with a table of contents and an index. This process allows regeneration of a website each time it is updated or when new pages are created.

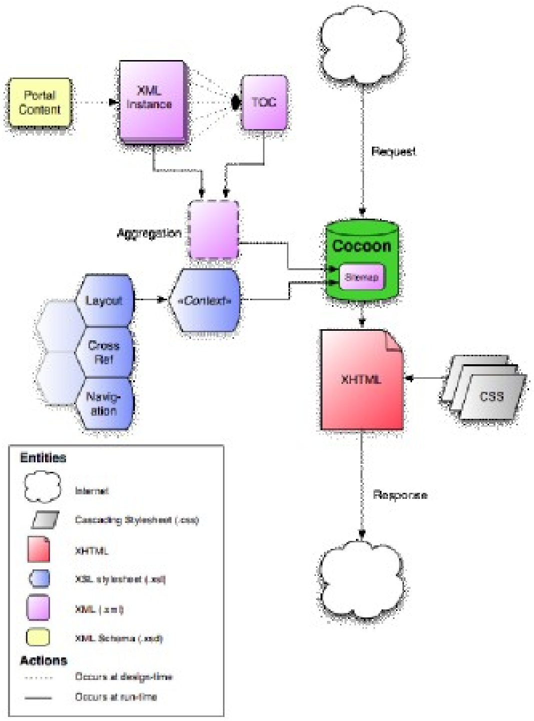


Figure 1. Center in a Box Version 1.0 Architecture

Figure 1, “Center in a Box Version 1.0 Architecture” (see [Cib1UG]) illustrates how Center in a Box handles a request and returns a styled XHTML page. Each of the instance documents resides in the file system on the web server. The site’s table of contents document (TOC) keeps track of each of these files and their associated metadata. When a request comes in, Cocoon uses the sitemap to match the requested URL with the appropriate XML document on the file system. To make cross-referencing and navigation between pages easier and more efficient, the sitemap aggregates the requested instance document with the table of contents. Cocoon transforms this aggregated document into a simple XHTML output using Center in a Box’s built-in XSLT stylesheet library. Finally, this simple XHTML document is styled at the client (most likely, but not always, a web browser) using CSS.

3. Center In a Box: An SOA Perspective

CIB 1.0 had the following problems which are solved by moving to a Service Oriented Architecture.

1. The XML pipeline in Cocoon held too much of the content manipulation rules.

2. Content creation could not be tested outside of the Cocoon environment, making authoring difficult.

3. The XML pipeline and the content model were too rigid, making it difficult to modify the structure of the presentation and content.

We sought to evolve Center in a Box version 1.0 from a single source publishing model to a Service Oriented Architecture with the intention of achieving a more flexible publishing model that is able to combine content components such as multiple person instances, or a person and an event either hierarchically or according to a graph model. This allows the combination of content irrespective of its hierarchical position. The content is then styled and then served in a variety of formats. For example, mobile users may not want content heavy feeds, so feed titles could be provided versus users requesting content from a desktop who want a more extensive version of a feed.

The main components of a Service Oriented Architecture are as follows:

1. Solutions that can extend or change on demand

2. Reusable services with well defined, published, and standard compliant interfaces.

3. A mechanism for integrating legacy applications regardless of their platform or language

Our application supports all the features of SOA listed above.The CIB 2.0 content model is flexible enough to support dynamic modification and updation of the website content. For example, it is easy to envision clicking on a person's name on a center's website and seeing a page of all the papers they have published, or the events they are involved in. The new flexibility of the CIB 2.0 content model and linkages by the incorporation of smallx pipelines allows the user of our system to generate content on the fly rather than try to predict all permutations of requests that a visitor might make.

Our architecture consists of a set of distinct reusable service components that can be recombined and repurposed in many forms to suit different services. University research centers are part of a larger organization, the University. Wide scale implementation of a CIB 2.0 standard would allow universities to develop applications that include, for instance, querying centers for event listings, or new publications by professors.

The services supported by our architecture such as website content creation, syndication, presentation, querying, and interfacing capabilities, are also sufficiently general to be extended to non-university based research centers.

While the architecture of CIB 2.0 is meant to replace that of an existing site, the flexible nature of the Atom components allows users to drop XHTML into an object, or keep whole HTML or XHTML content on their site and link to it. This allows a user to preserve existing content should they desire. We hope this will allow parallel content development, making it easier to implement CIB 2.0 without losing existing functionality during the transition. Also, it allows the use of applications that would not otherwise be supported by CIB 2.0, a capability that CIB 1.0 does not support.

And finally, the user interface is one of the important services that can be provided by a SOA. The application interface defines the required service parameters and the look and feel of the service result thus defining the functionality of a service independent of the underlying technology. Having a standard reusable and simple interface plays a major role in reducing the complexity of an application and increasing its ability to integrate with other applications. While we will initially implement a simple GUI based interface, such as a web form or an editor for CIB 2.0, but we can also envision more technologically savvy centers creating a dynamic service for updating that allows the users to push and pull content to and from the site.

4. Overview of Center In a Box Version 2.0 Architecture

To support a service oriented architecture, and create a more flexible framework for the users of CIB 2.0, we chose a graph-based model emphasizing bi-directional linkages between object components. We also allow substantial flexibility in the model by allowing the users to extend the existing data models through the use of XSD substitution groups.

Each primary and secondary category of data object is stored as an Atom news feed. Individual data objects are kept as Atom Entry objects. Linkages are maintained through the use of the link tag. The atopic application provided by the smallx [smallx] project gives us this functionality.

By allowing for extensions in the XML schemas, and providing standard templates for extensions, we allow the users to create schemas that fit their particular needs, making the data objects more or less rigid as they need to, while still retaining individual core identity. In addition, we provide users with the capacity to define their own linkages between data objects without breaking the model.

With the data model built on a flexible XML schema and an Atom news feed, we made considerable strides toward separating architecture and presentation. This was important to make our model Service Oriented. With the data model and architecture fully separated from individual presentation formats, the architecture becomes service neutral. Transformation into a website, a news feed, a blog, or some other service, becomes entirely separated from the data format of the website. This may sound paradoxical, since it is kept in a news feed format. However, our model envisions dynamically aggregating news feeds from the component structures.

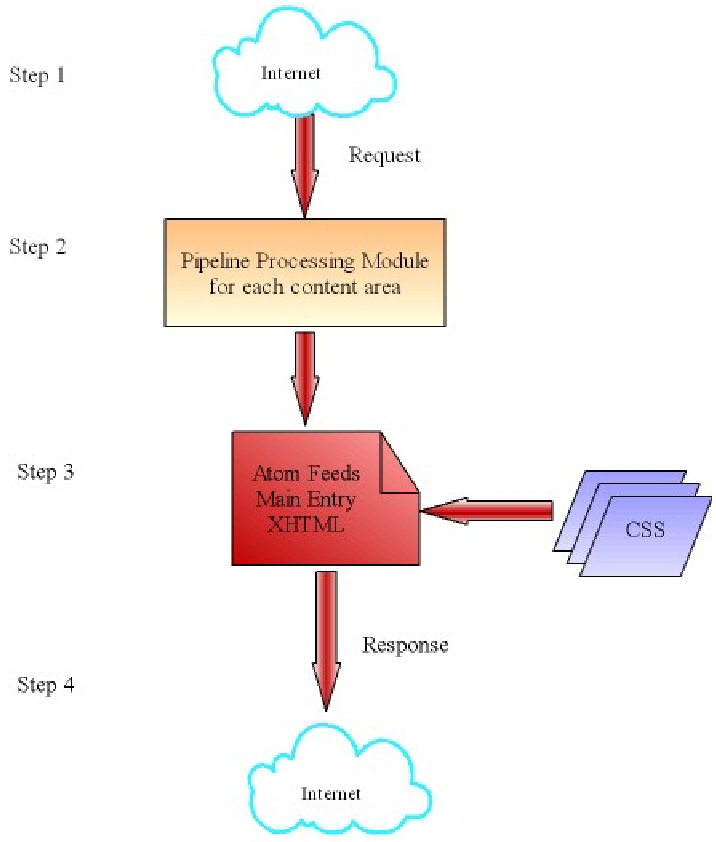

Step 1
Internet
Request
Step 2
Pipeline Processing Module for each content area
Step 3
Atom Feeds Main Entry XHTML
CSS
Response
Step 4
Internet

Figure 2. Center In a Box Version 2.0 Architecture

Figure 2, "Center In a Box Version 2.0 Architecture" illustrates the workflow of Center In a Box 2.0. The workflow is divided into four main steps. The first step starts when a request for a web page is made. The URL of the requested page is passed to Step 2 and the appropriate pipeline is called depending on the page requested. The pipeline aggregates and transforms the page and formats it as an Atom feed with a link to the CSS style sheet. This functionality was built into the atopic application [smallx]. We modified the code base for our needs and specific content. This is the most important step in the workflow as it creates both the content and the navigation paths based on metadata to assemble the final response page. This navigation defines the hierarchical architecture of the web page/site and the breadcrumbs are a reflection of the graphical aggregation of the pages. Figure 3, "Pipeline Processing Module of CIB 2.0" shows the expanded version of Step 2 in the workflow. In Step 3, the response document assembled in Step 2 is styled with the CSS style sheet linked to it. Finally the response document is returned to the user in Step 4.

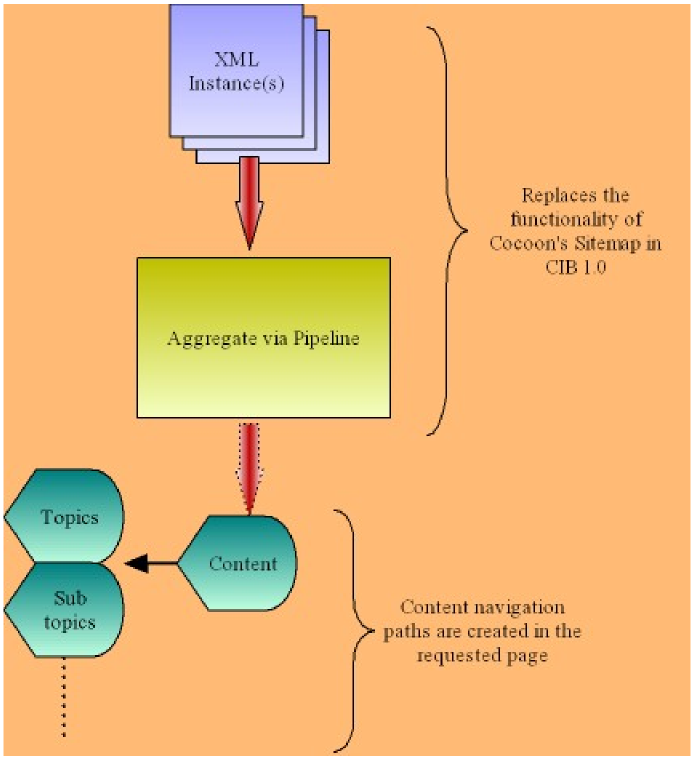

XML
Instance(s)
Aggregate via Pipeline
Replaces the functionality of Cocoon's Sitemap in CIB 1.0
Topics
Sub topics
Content
Content navigation paths are created in the requested page

Figure 3. Pipeline Processing Module of CIB 2.0

Figure 3, “Pipeline Processing Module of CIB 2.0” outlines the internal processes that occur in the Pipeline Processing Module of CIB2.0 workflow. The main function of this step is to convert the content contained in XML instance documents into Atom feeds. This allows easier editing of content components and allows content providers to deal with more familiar XHTML elements within a feed’s main entry element. The smallx Pipeline Processing Module in CIB 2.0 substitutes the functionality of Cocoon’s pipelines in CIB 1.0. Specifically, the CIB 2.0 pipeline performs multiple functions like aggregation of page components, transformation via XSLT, linking the CSS style sheet, and creating the navigation for each page along with a breadcrumb trail on the way out. The content can be arranged by topic, which is a hierarchical arrangement of the data, or by subtopics, which is a graphical arrangement of the data.

5. Data Sources and Modeling Methodology

We used a formal Document Engineering methodology to model CIB 2.0. The methodology included surveying websites, conducting interviews, and applying the step-by-step document engineering approach to create and implement our conceptual model.

5.1. Data Sources

We used the following data sources to harvest the candidate information components for the conceptual model of a research center.

5.1.1. Website Survey

We conducted a survey of 19 research center websites in the UC Berkeley domain and harvested candidate components from them to include in our content model. We chose the list of sample websites based on the following criteria:

1. The type of audience the site targets: Our sample included sites from both technical and non-technical academic disciplines. We defined technical as science and engineering, and non-technical sites included those centered on law or policy, social sciences, or community organizations.

2. The look and feel of the website:How attractive the website looks and the content management and presentation methodology. The purpose of this was to help determine how much emphasis we should put on presentation control versus structure and content.

3. The scope of the center's research: Our sample included research centers that perform broad research on fields ranging from IT, Engineering, Law, Politics to those focusing on specific topics like Tebtunis Papyri and Aging. This selection gave us an opportunity to identify and extract the common components from the most diverse University centers.

4. We harvested the websites at two levels of granularity to collect component information from a high level as well as the lowest possible level (most granular). The lowest level of granularity allowed us to identify all the atomic components present in the Center websites and the high-level harvest identified the common container components that constitute the content and structure of a research center website. The websites that we harvested are listed in Appendix 1.

5.1.2. Interviews

Due to time constraints, we were not able to interview as many people as we would have liked. However, we were able to interview one of the original beta testers for CIB 1.0, one of the creators, an researcher associated with CDE and expert user of the technologies used in CIB, and an advisor who was part of the final stages of CIB 1.0. We obtained their inputs about the strengths and weaknesses of CIB 1.0 and what they would like to be changed in the next version of CIB and their general approach towards content management and presentation.

5.1.3. CIB 1.0 XML schemas

A low level harvest of CIB1.0 XML schemas completed our data collection process. Harvesting CIB Schemas provided us with more structured information as compared to the harvest of websites and interviews. We were able to indirectly reuse the harvesting and analysis information that was utilized to create these XML schemas by harvesting the candidate components from the XML schemas themselves.

5.2. The Document Engineering Modeling Methodology

Throughout the modeling phase of this project the modeling team followed an approach called Document Engineering, developed by Robert Glushko and Tim McGrath. Document Engineering is defined as "a new discipline for specifying, designing, and implementing the electronic documents that request or provide interfaces to business processes via web-based services." [DocEng]

Document Engineering is a way to analyze information from diverse sources and merge it to create a single, unified data model. A Document Engineer has an "artifact-focused view of modeling," [DocEng] and begins by analyzing existing documents from businesses or entities that need to communicate with each other. This data is augmented by gathering information from other sources of requirements, such as the people who create or use the documents. A data model for the problem space is then created, which results in a set of reusable components that are usually expressed as XML schemas. This data model may then be used to build documents that all these entities can use to communicate. The use of this common data model allows different groups to exchange information in a loosely coupled manner, while still ensuring that all entities know exactly what the information means. Thus Document Engineering may be thought of as "a 'document-centric' version of the classical analyze - design - refine – implement methodology." [DocEng].

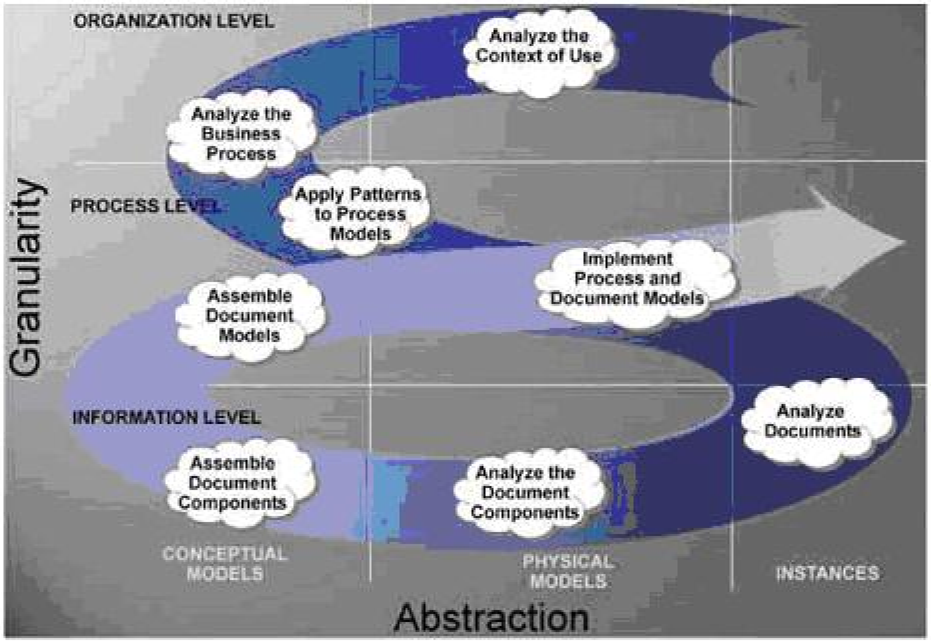

ORGANIZATION LEVEL
Analyze the Context of Use
Analyze the Business Process
PROCESS LEVEL
Apply Patterns to Process Models
Implement Process and Document Models
Assemble Document Models
Granularity
INFORMATION LEVEL
Analyze Documents
Assemble Document Components
Analyze the Document Components
CONCEPTUAL MODELS
PHYSICAL MODELS
INSTANCES
Abstraction

Figure 4. The Document Engineering Approach

Figure 4, “The Document Engineering Approach” (see [DocEng]) gives an overview of the Document Engineering approach as a path through the model matrix to carry out a set of analysis, assembly, and implementation tasks. The figure outlines the high level processes that define the document engineering methodology and also defines their order of execution. The processes are arranged in a matrix form that identifies their positions with respect to granularity and abstraction. We follow the phased approach outlined in Figure 4, “The Document Engineering Approach” for modeling CIB 2.0. The main steps of the process are as follows:

1. Analyze the Context of Use
2. Analyze the Business Process
3. Apply Patterns to Process Models
4. Analyze Documents
5. Analyze the Document Components
6. Assemble Document Components
7. Assemble Document Models
8. Implement Process and Document Models

6. Modeling CIB 2.0

We built upon the content model of CIB 1.0 and hence we had to incorporate the features and content model of CIB 1.0 while at the same time creating a more versatile content model in CIB 2.0. Our modeling process consisted of the following steps:

6.1. Business Process Analysis

The first step in creating a model for CIB 2.0 was to study the content model of CIB 1.0 and identify its strengths and weaknesses. This process coupled with the preliminary user and website survey that we conducted helped us to identify the scope and context of our project. We created business process worksheets and sequence diagrams to clearly identify, define and encode the context and high-level processes of our application.

6.2. Component Analysis and Assembly

6.2.1. Component Harvesting

We harvested the candidate components from the data sources discussed in Section 5.1 and created a candidate components table that listed the components, their basic data types and descriptions.

We began this process by harvesting the candidate components from each website that had been selected for analysis. At this stage we solely focused on the atomic meaning of the components and did not take into account their structural arrangement and hierarchy on the website. This task took on average about 2 hours per website.

6.2.2. Analysis

6.2.2.1. Analysis of Candidate Components

We did a comprehensive analysis of the data obtained from the harvesting the websites as well as the CIB 1.0 XML schemas. We created a component occurrence frequency chart, a part of which is shown in Figure 5, “Candidate Components Occurrence Frequency in sampled websites” to identify the frequency of component occurrence in the various Center websites. This chart helped us to identify the common patterns of occurrence of certain components such as “About,” “People,” “Publications,” etc. that occur in many research center websites and also gave us an opportunity to identify any pre-existing patterns of component occurrence.

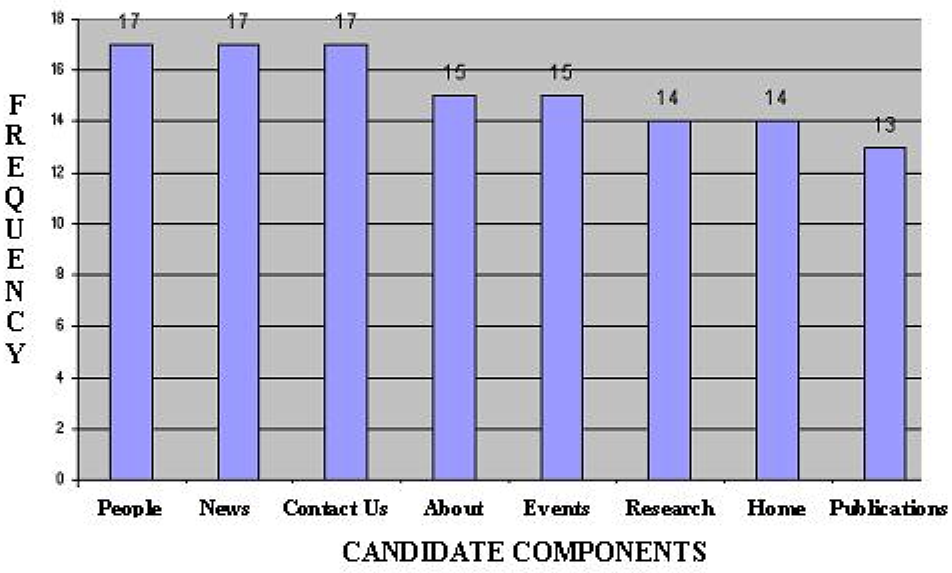


Figure 5. Candidate Components Occurrence Frequency in sampled websites

6.2.2.2. Analysis of Interview data

Our interviews also yielded interesting technical and end user perspectives about the problems with CIB 1.0 and provided valuable inputs on how our CIB2.0 can be modeled and developed. We extracted the following useful information after the analysis of our interview data.

1. The user interface should be model-based, highly intuitive, and easy to use.

2. Authoring, maintenance, and the import of legacy content need to be simplified.

3. The content model needs to be extensible and should permit repurposing and recombination. For instance, people can have publications, or Centers can have publications, or both, and the publications appear in different parts of the site. A mapping mechanism is essential to keep track of related data.

4. The Application should not have a steep learning curve, enabling even non-technical users to use the application.

5. CIB 1.0 Schemas (see [Cib1S]) are too rigid , and complex. XML schemas should be more extensible, easy to expand upon, and more atomic.

6. CIB 1.0 XML schemas tried to model too much, leaving no room for user extension and component modeling.

7. Good XML schema-aware tools should be made available for content creation.

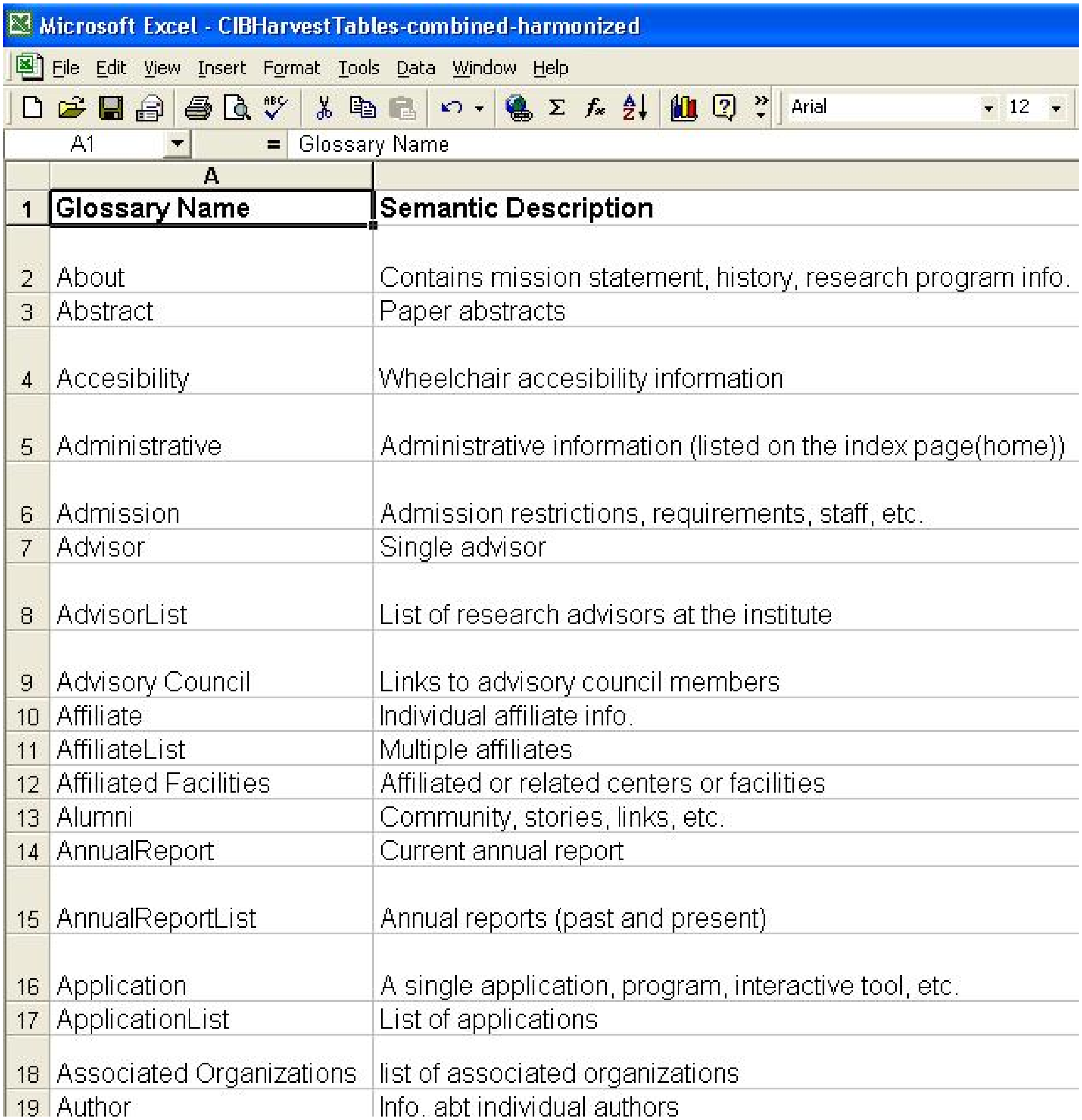
8. CIB2.0 should be able to import legacy HTML, XHTML, and other content.

From our interviews, we concluded that extensibility was an important requirement and that we needed simpler user interfaces to encourage the wider adoption of our application.

6.2.2.3. Component Aggregation and Harmonization

Based on the analyzed information in the previous step we aggregated and harmonized the components by removing redundant components and identifying naming irregularities. This is a very important step to identify semantic term conflicts and vocabulary problems described in Section 7. Component aggregation and harmonization ensures that all the possible meanings of a specific component are identified and its relation with other components is explicitly recognized.

We collected more than 350 data elements from the 19 Center websites. First, we merged all the individual spreadsheets into one master spreadsheet, which is referred to as the Table of Candidate Content Components. Next, we ensured that all elements had been assigned a glossary name. Elements that only appeared in one website were usually assigned a Glossary Name that was the same as their website component name, as were elements that appeared in more than one website but had the same element name. During this process we ensured that elements that were synonyms (different name, same meaning) had the same Glossary Name, and elements that were homonyms (same name, different meaning) had different Glossary Names. We then sorted the entire spreadsheet containing the data elements by Glossary Name. This spreadsheet, a part of which is shown by Figure 6, “Consolidated Table of Content Components”, then became our Consolidated Table of Content Components and contained close to 180 unique data elements.



	A	
1	**Glossary Name**	**Semantic Description**
2	About	Contains mission statement, history, research program info.
3	Abstract	Paper abstracts
4	Accesibility	Wheelchair accesibility information
5	Administrative	Administrative information (listed on the index page(home))
6	Admission	Admission restrictions, requirements, staff, etc.
7	Advisor	Single advisor
8	AdvisorList	List of research advisors at the institute
9	Advisory Council	Links to advisory council members
10	Affiliate	Individual affiliate info.
11	AffiliateList	Multiple affiliates
12	Affiliated Facilities	Affiliated or related centers or facilities
13	Alumni	Community, stories, links, etc.
14	AnnualReport	Current annual report
15	AnnualReportList	Annual reports (past and present)
16	Application	A single application, program, interactive tool, etc.
17	ApplicationList	List of applications
18	Associated Organizations	list of associated organizations
19	Author	Info. abt individual authors

Figure 6. Consolidated Table of Content Components

6.2.3. Document Assembly Model

The final step in our modeling process was to assemble the Center website document components to identify the high-level container components and the patterns of information flows between them. We had to make a choice between a hierarchical and a graphical representation for representing information flows between the high-level elements.

We concluded that a rigid hierarchical structure is not well-suited to the content management model for websites. This was an unexpected finding, because most center websites embody a hierarchical structure. But the centers can differ substantially in the choice of hierarchy, and two sites with the same content can follow exactly opposite hierarchies.

For example one website may have the component” People” as the container element that contains the component “Publications”. Another website may have” Publications” as the container element that contains the component “People” to arrange the publications according to the authors. To accommodate this variety, we chose a graphical representation showing bi-directional information flows as the content model of our application.

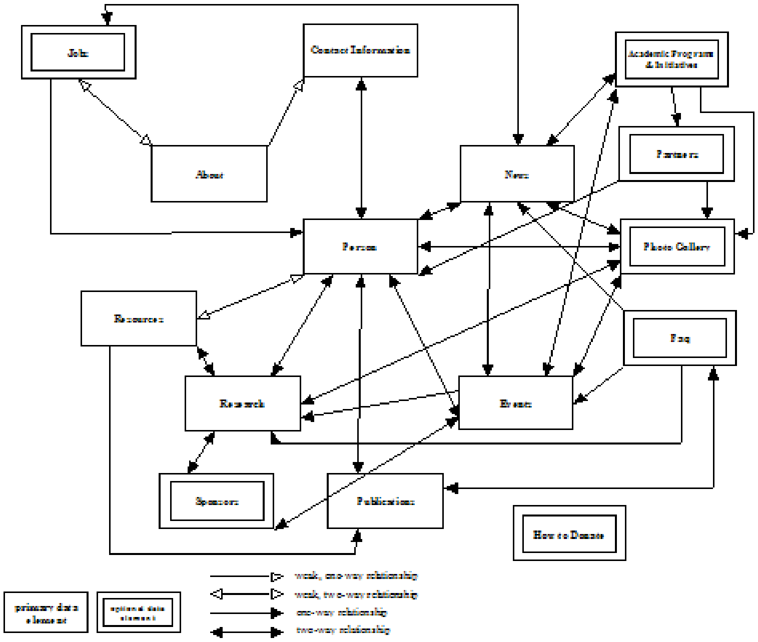


Figure 7. CIB 2.0 Document Assembly Model

Figure 7, “CIB 2.0 Document Assembly Model” shows the Document Assembly Model of CIB 2.0. Based on our analysis of the candidate components and our interview data, we divided our components into two groups, primary components and optional components. Primary components are the components that form an essential part of CIB 2.0’s content model. These components were present in more than 90% of the websites that we harvested and encode the minimal information that any research center website should contain. These components must be present in all Center websites encoded using our application. The primary components include elements like “About,” “People,” and “Publications” and are represented in the boxes with single borders.

The optional components are not essential parts of a Center’s website content. They consist of elements like “Jobs” and “How to Donate” that are not present in all center content models and are represented in the boxes with double borders.

Finally the solid single and double-headed arrows represent the uni-directional and bi-directional relationships between the components respectively. Hence the document assembly model represents the information flow hierarchy. We represent these information flows as “relationships” in our model. The solid double-headed arrows represent the bi-directional flow of information with no clear hierarchical structure. When two components are connected by a solid double-headed arrow, both components can act as containers for each other. The solid single-headed arrows represent a uni-directional containment. The hollow single-headed arrows represent a weak unidirectional containment relationship and the hollow double-headed arrows represent a weak bi-directional relationship wherein the components may be related to each other in certain but not all websites.

6.2.4. Important Modeling Lessons

6.2.4.1. How to identify the component harvesting chaff

Harvesting is more difficult than it looks. Two harvests of the same sources can look very different, especially if the two harvests are done at different levels of granularity. Very fine-grained harvesting can yield the lowest level document elements. These are the elements that cannot be further decomposed without losing all semantic meaning. For instance, an element “First Name” to denote the given, as opposed to family, name of a person, could contain the value John. At that level, John still has some meaning, though not much. Dissecting John further, to get the letters of the name, destroys the semantic meaning of the element. On the other hand, harvesting at a coarser level can yield elements that may eventually turn into assemblies of the low level items. Without those assemblies, the low level items have little semantic value. For instance, a “First Name” element is usually associated with a “Person” assembly. However, without a low level harvest, the Person element may not get all components it needs.

It is also difficult to know when to stop harvesting. We set out to harvest a set number of artifacts groupings, web pages from 19 sites, and the XML schemas from CIB Version 1.0. Harvesting half the number of websites would have yield the same Document Assembly Model. However, we did not know that until we began the harmonizing of the data elements. Ideally the process of harvesting can be considered to be complete when the new harvests do not yield any more new components.

6.2.4.2. The Art of Content Modeling

Modeling is an art, not a science. Even using the same methodology and the same initial data, people will select different components to harvest, and different approaches for the document assembly. They may also disagree on how much harvesting to do or how the relationship is defined between the components. We solved this problem through extensive brainstorming and taking a general consensus but there is always a potential for conflict in a less unified group, or between parties with different stakes in the project. On the other hand, as in our case, multiple individuals addressing the same problem can create a richer model.

7. Issues Identified by Our Modeling Analysis

7.1. Structural Stability vs. Design Flexibility

Though most research centers could benefit from having a stable content model for their websites, most had unique requirements that made a rigid content model unusable. This was also true of their presentation model. Walking the fine line between providing a stable and useful content model that is common for all types of research centers and at the same time supporting the ability to customize and individualize the websites according to a specific center’s needs was one of the most challenging tasks we encountered.

We addressed this issue by introducing the use XML Schema substitution groups into our content model in order to give the users of CIB the flexibility to change from generic to more specific terminology to describe their website content. Using substitution groups, "XML Schemas provides a more powerful model supporting substitution of one named element for another." [xml-schema] Also the evolution of CIB from a hierarchical to a graphical model also greatly enhances the customization of the structural organization of website components according to the requirement of the individual Centers.

7.2. Different Levels of User Technical Expertise

The modeling team's survey of campus research center websites during the modeling process revealed that websites on the UC Berkeley campus exhibit a wide range of technical complexity. Some websites exhibit a high level of complexity with a number of images, videos and other plug-ins. Others are static HTML pages that are a simple listing of the main components of the research center. Some lack even a rudimentary HTML site because of a lack of available resources and skill to create them. CIB 2.0 aims to meet the needs of as many of these different types of users as possible.

7.3. The Vocabulary Problem

The problem of finding a common term that means the same thing to all people is an old one. Furnas et al. documented the degree to which people disagree on the choice of words they use to name, describe, index things in a wide variety of contexts. (E.g., two people are likely to assign the same single word to things only 5-15% of the time). [VocabularyProblem] In such a scenario when we are building a common content model for a large variety of centers (ranging from research on pure sciences, social sciences, management etc), the occurrence of semantic term conflicts is inevitable.

We tried to address this problem by identifying common patterns in the various center websites and then dividing them into groups based on these patterns. Then we picked candidates from all the groups for our study sample and harvested the candidate components and compared the results to identify the semantic term conflicts.

8. Moving toward a more flexible architecture

Although version 1.0 of CIB met many of its original goals, it required significant space (at least 30 megabytes to run Cocoon) and its complex stylesheets are difficult to customize. We conducted a simple usability evaluation of CIB 1.0 that tested simple tasks common to website construction and maintenance, including content modification and structural changes. While basic tasks such as updating content and adding a few leaf nodes to the hierarchy are conceptually simple, they are cumbersome and require the use of direct XML editing. Anything more complex, such as changing the navigation to add another layer to the hierarchy is too difficult for most users.

Our immediate objective became to create a version of the CDE website that would allow developers and content providers to more easily update, add and delete content, or extend the architecture of the site according to their specific requirements. Additionally we sought to make it easier for a site built on CIB technology to be deployable anywhere. This required a change from Cocoon to another framework.

The smallx infoset and pipelining technology is a library and set of tools that was developed to process XML [smallx]. Smallx is capable of streaming documents and then processing those documents using pipelines. Pipelines allow the processing steps to be chained together to accomplish a task or set of tasks. We used smallx to apply pipelines to perform the tasks currently performed by Cocoon to process XML documents. We defined the same tasks using smallx pipelines as were originally defined in Cocoon's sitemap.

The CDE site is now based on the atopic application available from the smallx project. It uses the smallx XML pipeline technologies to aggregate each XML page with the table of contents, apply the appropriate transform, and output XHTML as requested. Specifically, a smallx pipeline was created for each of the core content areas. Each pipeline aggregates the designated XML page with toc.xml, transforms the resulting page with the appropriate XSL stylesheet, and outputs a XHTML document. It is also possible to easily modify the pipelines to output RSS or PDF. This increased flexibility and made it a bit easier to accomplish simple site maintenance tasks such as editing content, adding stylesheets and modifying the architecture, The smallx pipelines effectively substitute all the functionality of Cocoon’s pipelines. But we still needed to handle the URI mapping that sitemap was doing with its match blocks.

We used a urlrewrite filter [UrlRewrite] for the URI mapping. The urlrewrite filter takes care of determining which components to call and which resources to pass to these components based on the request URI. We configured the filter to abstract the URLs so that they remain clean no matter what the underlying technology or framework. The urlrewrite filter uses an xml file called rewrite.xml (that is present in the WEB-INF directory) for configuration and uses libraries from Apache Jakarta (Commons Digester and Oro). [UrlRewrite]

Using both smallx and urlrewrite, we are able to run the CDE site in Tomcat without Cocoon. We can easily update or change the content and architecture and deploy the site in multiple locations. We can make the changes using CIB2.0

in 1/5th of the time it took to make changes in the Cocoon based CIB 1.0. Our next step was to make CIB2.0 flexibile and simple so that content could be combined easily and flexibly using a graph model rather than a strict hierarchy.

8.1. Moving Toward a Service Oriented Architecture

Although the CDE website built using the CIB2.0 architecture is more flexible and more portable than the CDE website built using CIB1.0, there was more work to be done to achieve the level of flexibility and usability necessary to achieve our final goal of a SOA and the optimum level of appeal to users of CIB. Recreating the XML schemas using substitution groups would allow content providers to edit using familiar XHTML elements rather than having to deal with XML. Furthermore, content could now be incorporated during a transform or documents could be aggregated using a pipeline and the aggregate then transformed and output on the fly. We however, still needed to be able to aggregate our content as news feeds and present different view of the site content. The code base provided by the atopic application [smallx], gave us the flexibility we needed to combine components and build navigation to them according to our graphical model.

9. Atom Architecture

9.1. Why Atom?

Once we developed our content model, we became concerned that it would be difficult to create an architecture that would support the flexible nature of our application. We needed an architecture that supported the following characteristics:

1. Flexible user defined relationships between content items
2. Dynamic, user supported XML schema creation
3. Two way and one way relationships between content items
4. Type extension
5. Content creation and editing from within the application and through other applications
6. Dynamic reassembly of the content presentation by user and by client in multiple formats

XML was an obvious choice over a relational database or HTML/XHTML content architecture, since it is both flexible, and content driven, not presentation driven. Since we wanted our users to be able to create content models of their own through type extension, we felt that something that could use the XML Schema language would allow us to help the user control the content model, but still allow for a wide range of default constructs. Further, this would allow users with more technical knowledge to work with only a text editor if required.

Next we had to decide if we wanted a plain XML format or something specialized. Plain XML is highly transformable, and can be written with all our desired characteristics. In addition, it has the advantage of familiarity to many of our technically oriented users. On the other hand, we decided that it will be useful to support at least one syndication format. Creating content in a syndication format would give us the advantage of having the data already there. It would be fairly simple to transform from a syndication format into a plain XHTML or XML format.

In addition to ease of use considerations, we had to consider ease of extensibility versus the applications need to restrict users. In the end, we felt that a syndication format, and particularly Atom, would provide some built in restrictions on users by having a standard format, which XML would not. This would allow us to give some flexibility, but not allow the users to stray so far away from the content model that transforming to a syndication format or some other atomic format would be difficult.

9.2. Atom vs. RSS

The Atom versus RSS debate is well covered elsewhere (for instance at [RSSvAtom]). The arguments that most convinced us to use ATOM were the following:

1. Atom can carry well-formed XML, XHTML, or escaped HTML.

2. Atom can be used in a Atom Feed or a stand-alone Atom Entry element, which is useful for transporting via a web-service.

3. Atom is still under development, while RSS is frozen and it appears there will be no future versions. Admittedly, we liked the idea of being on the cutting edge.

We also felt that as Atom requires, the use of well-formed structures, and RSS does not though it supports them. Hence it would be comparatively easy to transform an Atom feed into an RSS. This is important as RSS is still widely used, and Atom is not yet widely deployed.

9.3. The XML schemas

Once we had decided on Atom, it was fairly easy to create the XML schema for each of the objects.

As shown by Section 9.3.1, The <link> tag in Atom allows us to specify the type of relationship between objects. Also by making the http://cde.berkeley.edu/Vocabulary/CIB/2005/2/0 namespace objects final, we create a core of data objects that cannot be extended, so that the application can rely on their presence.

9.3.1. CIB 2.0 XML schema Example

The examples below from our CIB XML schema shows the elements we use in our CIB feeds.The following is a snippet from the cibTypes schema. The cibTypes schema encodes the metadata used to create the navigation of each page. Specifically the title, id and subtitle elements are used in the dynamic creation of the navigation for the whole site

Example 1. cibTypes Schema Example

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified" targetNamespace="http://www.w3.org/2005/Atom"

  xmlns:t="http://www.milowski.org/2005/atopic"
  xmlns:xhtml=http://www.w3.org/1999/xhtml
  xmlns:a=http://www.w3.org/2005/Atom
  xmlns:cib="http://cde.berkeley.edu/Vocabulary/CIB/2005/2/0" >
<xs:import namespace="http://www.milowski.org/2005/atopic"
  schemaLocation="t.xsd"/>
<xs:import namespace="http://www.w3.org/1999/xhtml"
  schemaLocation="xhtml.xsd"/>
<xs:import namespace="http://cde.berkeley.edu/Vocabulary/CIB/2005/2/0"
  schemaLocation="cib20BaseTypes.xsd"/>
<xs:element name="feed">
<xs:complexType>
<xs:choice minOccurs="0" maxOccurs="unbounded">
<xs:element ref="a:title"/>
<xs:element ref="a:author"/>
<xs:element ref="a:entry"/>
<xs:element ref="a:id"/>
<xs:element ref="a:link"/>
<xs:element ref="a:subtitle"/>
</xs:choice>
<xs:attribute ref="t:navigation"/>
</xs:complexType>
</xs:element>
```

Example 2. PersonType Example

The snippet below encodes the details of the PersonType element. The PersonType contains specific elements like name,email and address that encode information that is specific to a person.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:cib = "http://cde.berkeley.edu/Vocabulary/CIB/2005/2/0"
  elementFormDefault="qualified"
  targetNamespace="http://cde.berkeley.edu/Vocabulary/CIB/2005/2/0">
<xs:element name="Person" type = "cib:PersonType"/>
<xs:complexType name="PersonType">
<xs:sequence minOccurs="1" maxOccurs="1">
<xs:element name="Name" type="cib:NameType" maxOccurs="1"
  minOccurs="1"/>
<xs:element name="ContactInfo" type="cib:ContactInfoType"
  maxOccurs="1" minOccurs="0"/>
<xs:element name = "ProfessionalTitle" type = "xs:string"/>
</xs:sequence>
</xs:complexType>
<xs:complexType name="NameType">
<xs:choice maxOccurs="1" minOccurs="1">
<xs:sequence maxOccurs="1" minOccurs="1">
<xs:element name="GivenName" type="xs:string" maxOccurs="1"
  minOccurs="1"/>
<xs:element name="MiddleName" type="xs:string" maxOccurs="1"
  minOccurs="0"/>
<xs:element name="FamilyName" type="xs:string" maxOccurs="1"
  minOccurs="1"/>
</xs:sequence>
<xs:sequence maxOccurs="1" minOccurs="1">
<xs:element name="FullName" type="xs:string" maxOccurs="1"
  minOccurs="1"/>
</xs:sequence>
</xs:choice>
</xs:complexType>
<xs:complexType name="ContactInfoType">
<xs:choice maxOccurs="unbounded" minOccurs="1">
<xs:element name="EmailAddress" type="cib:EmailAddressType"/>
<xs:element name="PhoneNumber" type="xs:string"/>
<xs:element name="Address" type = "cib:AddressType"/>
<xs:element name="Department" type = "xs:string" />
<xs:element name="WebPage" type="xs:anyURI"/>
</xs:choice>
</xs:complexType>
```

10. Conclusion and Future Work

In this paper, we presented a case study that outlines the conversion of a static content management system to service oriented architecture. We have created an architecture that supports reusability and repurposing of content components while at the same time providing a robust and stable underlying structure. We have also highlighted the advantages of using a syndicated format for encoding a SOA. We can combine content according to a hierarchy or according to topic (a graph model) and can present multiple views of the same data allowing us to serve up content in a format appropriate for a mobile device or browser.

One might easily ask, what kinds of services do university research centers need to provide? If there is simple content syndication and a web page, is that not enough? Our answer is that yes, of course that is enough.

However, we feel that our new model provides two major benefits immediately

The flexibility of the model allows for more dynamic creation of the website, and modification of content. For instance, it is easy to envision clicking on a person's name on a center's site and seeing a page of all the papers they are involved in, or the events they are leading. The new flexibility of the data model and linkages, coupled with dynamic pipeline creation, allows the user to generate content on the fly rather than trying to predict all permutations of requests that a visitor might make.

University research centers are part of a larger organization, the main University. Wide scale implementation of a CIB 2.0 standard would allow the university to develop apps that include, for instance, querying centers for event listings, or new publications by professors.

In addition, we view the Service Oriented Architecture as a benefit to user interface development. While we will initially implement a GUI based web form or editor, we can also envision more technologically savvy centers creating a dynamic service for updating that allows the users to push and pull content to and from the site.

Future work will include creating a model-based user interfaces that will be data driven and allow content providers to add, modify, or delete content using a user interface service. In addition, we will build in backend support for databases to make our application more adaptable to different kinds of backends.

A. Data Sources

The websites that we harvested are as follows:

1. California Biodiversity Center http://cbc.berkeley.edu/
2. Center for Entrepreneurship & Technology http://cet.berkeley.edu/
3. Chess: Center for Hybrid and Embedded Software Systems http://chess.eecs.berkeley.edu/
4. Center for Integrative Planetary Science http://cips.berkeley.edu/ [http://cips.berkeley.edu/]
5. Center for Research and Education on Aging http://crea.berkeley.edu/
6. Center for Studies in Higher Education http://cshe.berkeley.edu/
7. Engineering Systems Research Center http://esrc.berkeley.edu/
8. Institute of East Asian Studies http://ieas.berkeley.edu/
9. Institute of European Studies http://ies.berkeley.edu/
10. Institute of Human Development http://ihd.berkeley.edu/
11. Berkeley Nanoscience and Nanoengineering Institute http://nano.berkeley.edu/nanosite/
12. Center on Politics http://politics.berkeley.edu/
13. The Miller Institute for Basic Research in Science http://socrates.berkeley.edu/%7E4mibrs/

14. The Center for the Tebtunis Papyri http://socrates.berkeley.edu/%7Etebtunis/
15. Center for the Built Environment http://www.cbe.berkeley.edu/
16. Center for Information Technology Research in the Interest of Societyhttp://www.citris.berkeley.edu/
17. Human Rights Center http://www.hrcberkeley.org/
18. Earl Warren Legal Institute http://www.law.berkeley.edu/cenpro/earlwarren/
19. California Partners for Advancing Transit and Highways http://www.path.berkeley.edu/

Bibliography

[Atom] *ATOM Syndication Format* M. Nottingham, R. Sayre2005 http://www.atompub.org/2005/08/17/draft-ietf-atompub-format-11.html

[Cib1UG] *Center In a Box 1.0 User's Guide* 2003 http://groups.sims.berkeley.edu/CDE/releases/CIB/

[Cib1S] *Center In a Box 1.0 Schemas* 2003 http://groups.sims.berkeley.edu/CDE/releases/CIB/

[Cocoon] *Apache Cocoon* 2005 http://cocoon.apache.org/

[DocEng] Glushko,R.J., McGrath,T., *Document Engineering: Analyzing and Designing Documents for Business Informatics and Web Services*, MIT Press, 2005

[RSSvAtom] *Rss20AndAtom10Compared*, on the Atom Project Wiki, http://www.intertwingly.net/wiki/pie/Rss20AndAtom10Compared

[smallx] *Smallx XML Infoset and Pipeline* 2005 http://smallx.dev.java.net

[UrlRewrite] *URLRewrite Filter*, http://tuckey.org/urlrewrite , 2005

[VocabularyProblem] Furnas, Landauer, Gomez, Dumais, *The Vocabulary Problem in Human-System Communication*, Communications of the ACM 30(11), 964-971, 1987

[xml-schema] *XML Schema Part 1: Structures Second Edition* Henry S. Thompson, David Beech, Murray Maloney, Noah Mendelsohn2001 http://www.w3.org/TR/xmlschema-1

Biography

Mano **Marks**
University of California, Berkeley
Center for Document Engineering, School of Information Management and Systems [http://cde.berkeley.edu/]
102 South Hall
Berkeley
California
94720-4600
United States of America

Mano Marks is first-year master's student in the School of Information Management and Systems at the University of California, Berkeley. Before coming to SIMS he worked at a variety of non-profit jobs, managing all aspects of data collection, data retrieval, data analysis, data transfer and data reporting. Most recently, he was the Data Manager at Huckleberry Youth Programs. His research interests are process modeling, electronic document and data exchange, and information policy.

Chitra **Madhwacharyula**
University of California, Berkeley
Center for Document Engineering, School of Information Management and Systems [http://cde.berkeley.edu/]
102 South Hall
Berkeley
California
94720-4600
United States of America

Chitra is a second year Masters Student at UC Berkeley's School of Information Management and Systems. She holds a Masters degree in Computer Science from National University of Singapore. Chitra is a Researcher at the Center for Document Engineering, UC Berkeley. For the past four years, she has been actively involved in research on XML based metadata transformations. She is interested in doing research on the techniques to design and develop XML based Knowledge Management Systems that incorporate a Service Oriented Architecture.

Christine **Jones**
University of California, Berkeley
Center for Document Engineering, School of Information Management and Systems [http://cde.berkeley.edu/]
102 South Hall
Berkeley
California
94720-4600
United States of America

Christine Jones is currently a first year masters student at the School of Information Management Systems (SIMS) at UC Berkeley. She earned her degree in Political Science from UC San Diego, and worked in The Scripps Institute of Oceanography Library while earning her degree. She has worked as a web developer as a contractor and in the corporate world, and currently works at UC Davis, Shields Library in the Systems department as a programmer.

Christine's research interests are Document Engineering, Digital Libraries, and Databases. As a programmer at UC Davis, she has been working on Digital Initiatives for the past two years providing support on encoding; creating best encoding practices and documenting them; working with SGML, TEI-Lite, XML, and XSLT to encode and transform collections; and is the administrator of the Digital Initiatives Website.

Robert **Glushko**
University of California, Berkeley
Center for Document Engineering, School of Information Management and Systems [http://cde.berkeley.edu/]

102 South Hall
Berkeley
California
94720-4600
United States of America

Bob Glushko is an Adjunct Professor at the University of California at Berkeley in the School of Information Management and Systems and is the Director of the Center for Document Engineering. He has twenty-five years of R&D, consulting, and entrepreneurial experience in information management, electronic publishing, Internet commerce, and human factors in computing systems. He founded or co-founded three companies, the last of which was Veo Systems in 1997, which pioneered the use of XML for electronic commerce before its 1999 acquisition by Commerce One. From 1999-2002 he headed Commerce One's XML architecture and technical standards activities and was named an "Engineering Fellow" in 2000. He has an undergraduate degree from Stanford, an MS (Software Engineering) from the Wang Institute, and a PhD (cognitive psychology) from UC San Diego.