Annotation Management in XML Content Management Systems, A Case Study

David **Price** <davidp@isogen.com>

Abstract

The creation and management of annotations on XML documents is sometimes a requirement for content management systems, even very simple systems. An annotation in this context can be defined as a short piece of information with three qualities: it is associated with a particular piece of content, it is created and viewed outside the normal authoring environment or tool, and it is stored separately from the annotated content. In XML documents, annotations are usually associated with a particular element. One use case for annotation is during a document review and revision process, although they are by no means limited to this. Although annotation of XML documents is in itself not difficult, the process becomes much more complicated when integration with document workflow, document versioning, and link resolutions become necessary.

This paper presents a case study of an XML annotation management system implemented on a content repository based around Documentum 4i. The annotation manager in this case was required to deal with all of the situations described above: versioned documents, integrated workflow, the need for annotations to be created outside the normal authoring environment, and the need to display linked documents during annotation. Also detailed in the paper are the design and capabilities of the system, pitfalls that were encountered during the construction of the system in both a design and development sense, and the compromises that were made in order to deliver the system.

Table of Contents

1. Introduction

The ability to add annotations to content is a requirement within many content management systems, often those in which there is a need for some kind of formal or informal review process. Our software development team was tasked with developing a lightweight and scalable annotation system that would be accessible via multiple interfaces and would integrate flexibly into the workflow of the client's Content Management System (CMS). Increasing budgetary constraints on the part of the client also put a premium on minimizing the time and resources required to develop and integrate the solution.

The development team consisted of three developers. Two team members were on the project from conception to deployment, and the third was involved in the middle of the project to assist with the rendering of the markup into

a human-readable form. The project as described took place over a period of two months in the fall of 2001, and was installed at the client site at the end of the development period.

2. Background

In order to better understand the implementation decisions made during this project, it is useful to first explore the problem of annotation and annotation management in a more general sense. Annotations in general are a specific subset of meta-information with three characteristics:

- It is applied at a finer granularity than most content management systems store files. In a system where content is managed at the document level, this distinguishes it from document-wide metadata. For XML systems in particular, element-level addressing for annotations is most common. While a few CMS offerings currently available do chunk documents at the element level[Astoria], most do not.
- It is created and viewed outside the normal authoring process, although not necessarily outside the authoring tool. This distinguishes annotations from any detailed metadata that might be added during authoring or by other authors in an authoring process.
- It is stored and managed separately from the content with which it is associated.

Annotation management has a potential use case in any system in which metadata is important, and there are a wide variety of real-world examples of annotation management systems already in existence. Applications include compiler information, geospatial information, document management systems, and issue tracking systems.[1] Several on-line documentation systems use annotations as well, particularly in the open-source community where a premium is placed on end-user interaction, information, and involvement with the product as opposed to central control of the product.[2]

XML lends itself particularly well to annotation thanks to the structure that it imposes on information, and the linking technologies that have been built around it. This makes the association of annotations with particular parts of the annotated information particularly easy. One of the use cases for XML annotation occurs within large content management environments that require a lightweight feedback mechanism to communicate detailed information regarding specific aspects of the content from one workflow stage to another, usually as a part of a review process.

3. System Requirements

The authoring system being used by the client was based around Documentum 4i[3] as a CMS and Arbortext Epic 4.2[4] as the primary authoring tool, with Solaris 8 as a target server operating system and Microsoft Windows NT and 2000 being the target for clients. Any additions to the system also had to be compatible with an ISOGEN-provided Link Management System (LMS)that had already been developed. [5]The client did not require any cross-platform capability beyond the targeted systems.

[1]BattleScape[BattleScape] and ArcGIS[ArcGIS] are examples of geospatial information systems that offer annotation features.

Documentum and Astoria are examples of document or content management systems that offer annotation functionality.

[2]An example of this is the PHP documentation system, seen at http://www.php.net/manual/en/

[3]Documentum is a popular large-scale content management system that includes versioning, workflow, and an interface that allows for extensive customization in Java and via COM.

[4]Arbortext Epic is an XML/SGML editor that is commonly used in technical authoring environments. It is customizable via a proprietary scripting language (ACL). Epic is relatively Java-friendly thanks to additions that have been made recently to the ACL language.

[5]The custom linking solution written by ISOGEN provided a version-safe method of linking between documents, enabled a where-used query functionality within the system, and guaranteed that every document in the system had a unique version-independent identifier. Although the linking solution had additional capabilities, none of them are relevant to a discussion of annotations.

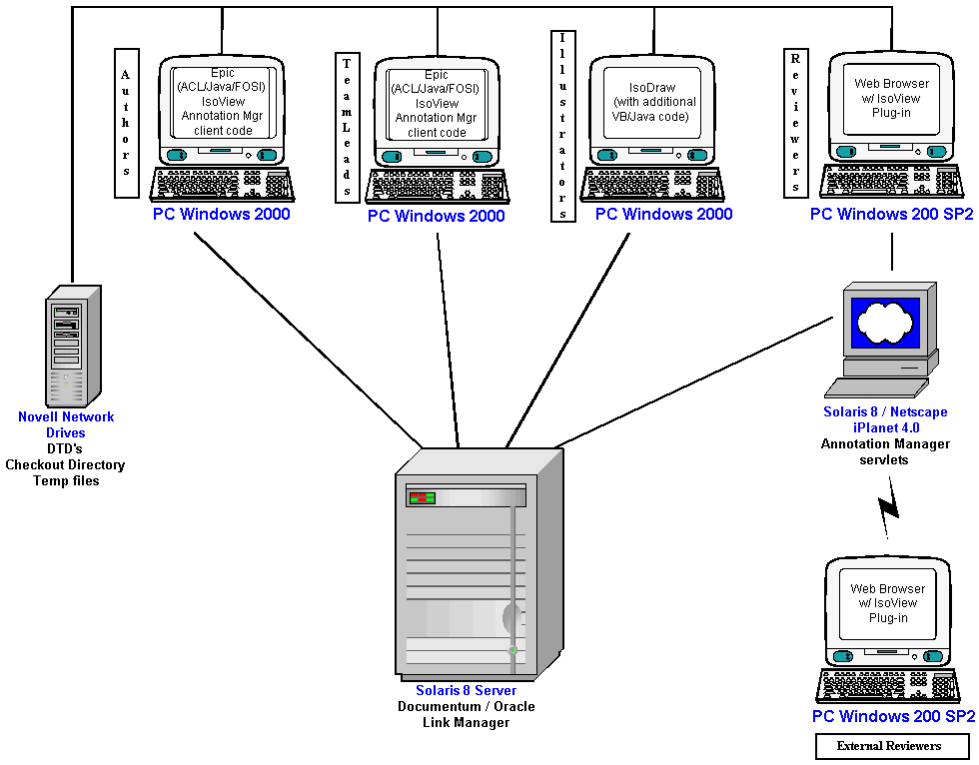

Figure 1. The general system environment for the annotation manager.

Although the system itself is not particularly complicated, the requirements it had to fulfill were involved:

- Interface integration with the Arbortext Epic product was required for authors and editors who would be receiving annotations.
- A web-based interface was required for reviewers for the creation and querying of annotations.
- The reviewers were not considered to be XML savvy, so in addition to creating and querying annotations, it was necessary to add something to render the authored XML into user-friendly HTML. Because the documents in question had an extremely complex DTD[6], this was a non-trivial portion of the design.
- The entire system needed to be flexibly integrated into the built-in workflow available through the Documentum CMS.
- The management of annotations needed to be compatible with the existing custom link management scheme and the versioning capabilities of Documentum.

Although Documentum 4 does have annotation capability, it was deemed insufficient for the needs of the client.[7] Thus, we set out to design a custom solution that would satisfy the requirements listed above.

4. System Design

For a generic annotation manager system such as the one described in the System Requirements Section 3section, there are only two hard problems that need to be solved:

1. Linking annotations to documents or portions of documents, and managing those links over time.
2. Integrating annotation management into the system workflow in an adequately flexible manner.

[6]Specifically, this was a DTD based on the IETM Class 5 specification.[IETM]

[7]This may be changing in the next version of Documentum, please see http://www.cmswatch.com/News/Article/?129 and Documentum's website at http://www.documentum.com/for more details.

The other problems involved in satisfying the requirements are significantly less difficult. Actually creating, storing, and manipulating annotations are relatively well-explored problems that can be solved with existing tools, as are rendering the annotatable XML documents for display and building user interfaces to manipulate the annotations.

In this sense, the problem of annotating documents can be seen as a subset of a more general problem, that of linking two pieces of information and managing those links. Once the linking problem has been solved, the creation and management of annotations outside of any system workflow falls into place rapidly. The workflow problem remains, but is less easily generalized.

With this in mind, it was realized that the custom link manager that had already been integrated into the system solved most of the difficult parts of annotation without any additional effort on the part of the team. Although a detailed discussion of it is beyond the scope of this paper, it is not necessarily difficult to build a link management system that will fulfill these basic requirements, depending on what tools are available and the CMS being used. There are three requirements that have to be met by a link manager in order to serve an annotation management function:

1. Index content units by a unique identifier that does not change across versions. This is essential for annotations to link consistently to a particular document.

2. Index content units by a second unique identifier that is guaranteed to change each time a document is versioned. This can be something as simple as a version number - it does not need to be unique across all documents, just within the version identifiers of this document.

3. In some cases, such as linking to information within a content unit across multiple versions, it may be necessary to assign unique identifiers to each annotatable item within the content unit, which must be changed whenever the content within that annotatable item changes. This was initially the case for this project, although later the functionality that required this was eliminated.

5. System Implementation

Java was determined to be the language of choice for implementing the project. Java was the easiest language with which to share code between the web application for reviewers, the integrated application for editors, and with the already existing CMS customizations that were written mostly in Java.

The system was divided into three packages, each addressing a separate group of requirements:

- **Annotation Manager Core** - the Core package is a set of interfaces, abstract classes, and implementations of those classes that provide an interface to basic annotation management functions such as annotation creation, modification, querying, and some basic document access functionality.

- **Annotation Manager GUI** - these are a set of Swing interfaces that use the Core package to provide access to annotations through a desktop GUI.

- **Annotation Manager Web** - this contains all of the Servlets and related code for allowing web based access for reviewers.

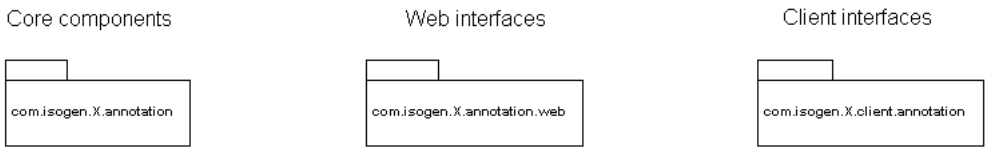

Figure 2. Annotation manager package listing.

5.1. Design and Implementation of Annotation Manager Core Package

The two driving principles behind the design of the core components for the annotation manager were a desire to separate interface from implementation as much as possible to create a portable implementation, and to make the interface as simple as possible. During the initial design phase, it was determined that a non-implementation-spe-

cific interface would allow us to develop the three major components of the system in parallel, further reducing the amount of time it would take the team to deliver the solution. The opportunity to provide a limited interface using a system other than Documentum, which can be complicated to integrate with, meant that limited testing on other components could be done before the core component was fully functional. In this case, the team developed a limited implementation that used Java DataBase Connectivity (JDBC) to access a Microsoft Access database for preliminary testing and development purposes. This prototyping allowed the team to sort out deficiencies in the interface itself quickly, before the more difficult work of creating the full Documentum implementation.

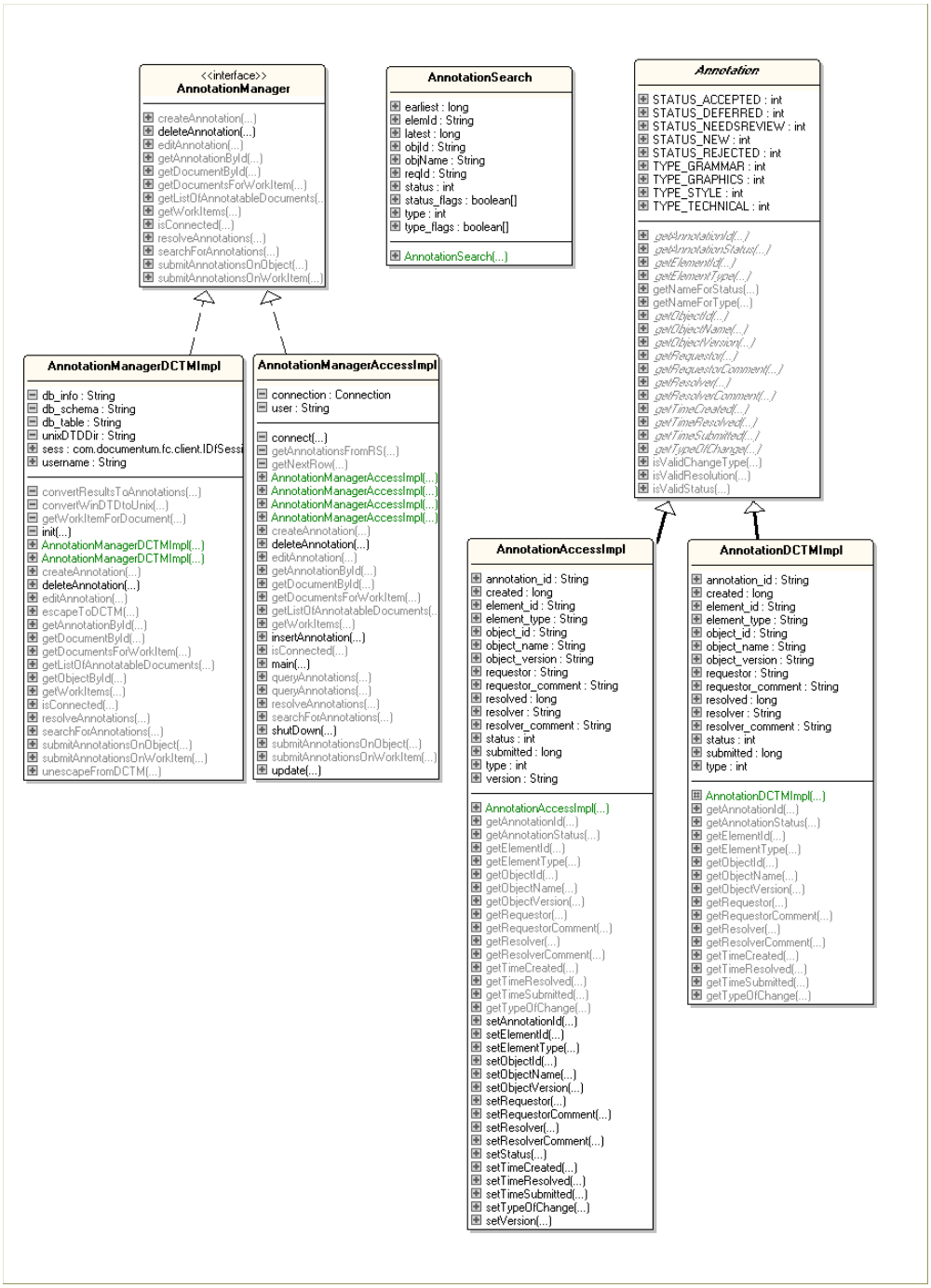

Figure 3. Class diagram of objects in the core package.

The implementation of the annotation management interface in both cases used a relational database to store annotations and manage them. Storing the annotations as XML within the content management system was considered, but was rejected for several reasons. Storing the annotations as XML was an unnecessary complexity with little apparent value, since there was no requirement to be able to access annotations independently of the tool.

Storing the annotations in a relational database had the advantage of being significantly faster than storing them as XML within the CMS, and made querying annotations much simpler. Because of the simplicity of the relational database structure and queries for annotations, it was also much faster to implement.

With a more generalized interface, the actual information contained within the annotation is not really important in a general sense, only for specific implementations. Unfortunately, that is less the case with this interface as certain aspects of Annotation objects were placed directly into the AnnotationManager interface.

The Core interface consists of three main abstract objects, implementations of those objects, and some miscellaneous helper classes and exceptions. The AnnotationManager interface contains all of the basic operations of annotation management. The Annotation abstract class defines basic fields that must be present in an annotation. The AnnotationSearch interface defines a basic interface to searching functions. This architecture had the potential to keep the AnnotationManager object itself relatively independent of the structure of annotations.

The problem that this architecture does not solve, at least not very well, is the problem of integrating workflow. This is the point in the design where the team ran up against the second hard problem in the construction of an annotation management solution, and suffered for it.

The original implementation of the annotation manager ended up with an interface heavily polluted by annotation specifics and workflow. Since the aim of the project was a custom annotation manager done quickly and cheaply, a decision was made to not refactor the interface.

Workflow ended up being the most damaging addition to the interface, and caused the most problems in actual implementation. The actual workflow required was quite simple in concept: Reviewers would receive a document in their inbox, make annotations on it, and promote it to the next stage of the workflow when finished. Authors would receive a document in their inbox with annotations attached. They would then be responsible for resolving those annotations and promoting the document to the next stage of the workflow when it was ready.

Implementing this workflow within the Documentum system was less than simple, particularly since the team was inexperienced with Documentum workflow at that time. The implementation of the system reflects this with added clutter that is too Documentum-specific to be as generic as we desired. Promotion methods and other workflow-specific concepts leaked into the AnnotationManager interface and were never factored out into a separate class.

5.2. Design and Implementation of Annotation Manager Swing GUI Package

The GUI package is a simple Java Swing interface that provides a limited front end for the Core classes. It has two integration points with the Arbortext Epic editor - one with the topbar menu where it is first called, and one with the editor itself so that the element annotated by the current annotation is selected in the main window.

5.3. Design and Implementation of Annotation Manager Web Interface Package

The basic structure of the web interface was made easier by the choice to use Java. Using servlets, adding a web interface on to the core packages was almost as easy as adding a simple Swing interface. What complicated things was the requirement to display documents to the reviewers as user-friendly HTML rather than raw XML. We decided to use a custom SAX[SAX 2.0] application to do this. We chose SAX over XSLT[XSLT] primarily because of speed. Raw SAX programs are faster in many circumstances than XSLT scripts, and speed was paramount because we did not want to implement the complexity of caching transformations and yet still needed to provide an acceptably quick transform on documents that were potentially very large.

A base class for the SAX transformer was created, and a child class of this was created for each document type that required transformation. The SAX-based transformer operated by calling a method using the element name the parser encountered as an index. Methods for rendering common elements were placed in the base class, and child classes could either use the parent method or override it when element formatting was different from doctype to doctype.

This method was very fast, but not as flexible as an XSLT-based solution. Had there been an opportunity, we would have liked to refactor this portion of the design to use XSLT rather than SAX for exactly this reason. As

the client requested changes in the way pages were rendered, it became increasingly difficult to maintain the custom SAX application. Allowing the client to alter the rendering themselves via XSLT, and finding another means to increase the speed of rendering, such as rendition caching, would be a more optimal solution.

6. Implementation Timeline

Work began on the annotation manager in mid-September of 2001 with the two original team members starting development work on the Core packages and web interface. Work on these proceeded in parallel. Around the first of October an additional developer was added to create the SAX transformation code used within the web interface. Development against the Documentum implementation of the core package, rather than against an Access-based mockup, began in mid-October round. The SAX transformation code was completed around the same time, although client requests would cause it to be tweaked throughout the construction of the system. At this point the team shrunk back to the two core developers.

An initial review of the system with the client was held on the 17th of October, and at that point the system was functional enough for a nearly complete demo. A second customer review took place on November 9th. Both of the reviews were remote demonstrations rather than client installations. Development plus testing were complete on November 19, and the system was installed at the client site a week later. In all, the development of the entire system took roughly two months for a team of two developers, plus another two weeks from a third developer for the SAX XML to HTML transformations.

Much of the development time was spent learning how to implement what we knew we needed to do within Documentum. Had we known what we do now, or if we were working with a CMS in which we had more experience, it is probable that another one to two weeks could have been saved.

7. Results and Conclusions

Adding a customized annotation management system to an existing CMS solution is not necessarily an extraordinarily difficult task, as long as the two fundamental problems of annotation management are kept firmly in mind while designing the system and the implementation of the system is kept as simple as requirements allow. The interface may not be extremely polished for the initial delivery, but can be refined later if the annotation manager architecture is designed in a flexible manner. Customized annotation management can add a significant amount of value to a content management system for a relatively small cost, particularly if a general link management solution is available or has already been integrated.

Bibliography

[IETM] Interactive Electronic Training Manual, MIL-PRF-87269, http://log.dau.mil/ietm-toc.asp

[Astoria] Astoria CMS, from Lightspeed, http://www.lspeed.com/products/ls_p_astoria/frameset.html

[SAX 2.0] Simple API for XML, version 2.0, http://www.saxproject.org/

[XSLT] XSL Transformations (XSLT) Version 1.0, http://www.w3.org/TR/xslt

[BattleScape] BattleScape geospatial data visualization environment, http://www.autometric.com/NEW/Products/Visualization/bs.html

[ArcGIS] ArcGIS Geographic Data Management System, http://www.esri.com/software/arcgis/index.html

Glossary

ACL	Arbortext Command Language
CMS	Content Management System
IETM	Interactive Electronic Technical Manual
JDBC	Java DataBase Connectivity
LMS	Link Management System

Biography

David **Price**
ISOGEN International, LLC
Dallas
United States of America
davidp@isogen.com

David is currently a Software Engineering Consultant for ISOGEN International. As an experienced Java and C++ developer, he has helped to bring several large customized content management and authoring systems into being in the telecommunications and defense industries. A deep appreciation of standards-based technologies, agile development techniques, good modeling, and really good coffee has resulted from his experiences.