An XML Based Knowledge Management System for e-Collaboration and e-Learning

Varun Gopalakrishna[1], Ashwin K Bhagavatula[1], Lih-Sheng Turng[2*]
[1]Department of Electrical and Computer Engineering
[2]Department of Mechanical Engineering
University of Wisconsin-Madison
Madison, WI 53706

ABSTRACT

This paper presents the development, key features, and the implementation principles of a sustainable and scaleable knowledge management system (KMS) prototype for creating, capturing, organizing, and managing digital information in the form of Extensible Markup Language (XML) documents and other popular file formats. It is aimed to provide a platform for global, instant, and secure access to and dissemination of information within a knowledge-intensive organization or a cluster of organizations through Internet or intranet. A three-tier system architecture was chosen for the KMS to provide performance and scalability while enabling future development that supports global, secure, real-time, and multi-media communication of information and knowledge among team members separated by great distance. An XML Content Server has been employed in this work to store, index, and retrieve large volumes of XML and binary content.

Keywords: Knowledge Management System (KMS), Native XML Databases, Search Utility, Extensible Style Sheet Language (XSL) Transformations.

INTRODUCTION

During the last decade, the Internet and the World Wide Web (Web) have greatly changed our daily lives and reshaped the landscape of business and commerce. In this information age, traditional pillars of economic power – capital, land, plant, and labor – are no longer the main determinants of business success. Companies are beginning to realize that their competitive edge actually relies on the brainpower or the "intellectual capital" of their employees and management. Knowledge management (KM) is the discipline by which organizations manage and re-use their enterprise-wide knowledge to gain a competitive edge [1,2]. When the right information is delivered to the right people at the right time, the intangible information transforms into knowledge based on which rational decisions can be made. This paper presents the development of a KMS prototype, which provides a framework to facilitate the creation, capturing, organization, and sharing of knowledge for various knowledge domains. The following sections provide an overview about the XML technologies involved and the specific implementation details of the KMS such as the three-tier architecture and main features of the system, a Web-based client Application Programming Interface (API), and the server's Component Object Model (COM) objects. The need for XML as a data notation system for such a system rather than HTML is also emphasized.

XML Data Formats

Knowledge management can be facilitated by the availability of a common, processable message format and mechanisms for establishing relationships between messages and their context which effectively embodies the data about the information itself. Being a generic representation of content, XML materializes the concept of "write content once, re-use at will," leading to the evolution of the next generation of XML application and Web Services [3].

Using XML as a notation system for the actual data formats means that substantive properties of the information are represented as a meta-data scheme encoded with the actual information. This would ensure that the primary data structure remains unambiguous. Therefore, not only can data be mined from backend systems but can be indexed for searching and can be personalized for each user accessing the data. The semi-constructed nature of data works well for integrating heterogeneous data and modeling varying sources, thereby making it suitable for general and intuitive knowledge representation. Implemented in conjunction with platform-independent protocols, XML facilitates exchange of data and ease of Web-based software development. It is designed to improve the functionality of the Web by providing more flexible and adaptable information identification. XML allows the flexible development of user-defined document types. It also provides a robust, non-proprietary, persistent, and verifiable file format for the storage and transmission of text and data both on and off the Web.

XML facilitates applications in which intelligent Web agents attempt to tailor information discovery to the needs of individual users and to distribute a significant proportion of the processing load from the Web server to the Web client. These are the fundamental features that could be exploited in the design and creation of a knowledge management system. With the inherent structure of XML data, the KMS is well suited to existing hierarchic classification of knowledge allowing incorporation of a high level of dependency and inheritance among knowledge.

Difference between HTML and XML

Early generalizations about XML have led many to believe that XML is just a method for extending HTML by adding new tags. In fact, XML and HTML exist in entirely different layers of markup technology. HTML is a tag language (conceptually, a markup language) – a set of standard delimiters with standardized meanings [4]. Dedicating a small set of tags allows users to leave the language specification out of the document and makes it much easier to build applications, but this ease is at the expense of severely limiting HTML in several important respects, such as extensibility, structure, and validation [5].

On the other hand, XML differs from HTML in three major respects, namely (1) customized tag and attribute names, (2) document structures can be nested to any level of complexity,

*Corresponding author

(3) XML documents contain Document Type Declaration (DTD's) or schemas for use by applications that need to perform structural validation.

While HTML will continue to play an important role for the content it currently represents, many new applications require a more robust and flexible infrastructure. XML leads HTML to XHTML. XHTML family document types are XML based, and ultimately are designed to work in conjunction with XML-based user agents [6].

Native XML Databases (NXD)

A Native XML Database primarily defines a logical model for an XML document. It stores and retrieves documents in accordance with this model. Any such reference model must include elements, attributes, parsed character data (PCDATA), and document order to be directly processed by applications such as Web Services without an additional translational layer. The NXD's preserve document order, processing instructions, comments, character data (CDATA) sections and entity usage. The NXD's also support features like transactions, security, multi-user access, programmatic API's, and query languages, which would be pivotal in the design and creation of any content management system.

With no dependence on a predetermined structure, such a content server can store and retrieve heterogeneous document types and is well suited to applications where data varies in content and structure. Hence such a database is used to build a versatile prototype. However any application that needs data in a different format must parse the XML rendered by the NXD.

There are several reasons for storing data in an NXD. In particular, the XML content destined for information retrieval applications is semi-structured rendering it inefficient for mapping to a relational database. It enables higher retrieval speed (since storage strategies use physical pointers between parts allowing the documents to be retrieved without any logical joins). It also exploits XML specific capabilities (support for XML queries).

These relational database models are ill suited for rich and hierarchical XML content due to the inherent storage as a Binary Large Object (BLOB) which leads to high usage of memory, higher error rates and, in turn, lower processor performance. Storing and indexing XML documents in their native form eliminates these problems and increases the performance of the applications that depend on the data. XPath could be used for querying collection of documents. However, it is not intended to be a database querying language since it lacks grouping, sorting and cross document joins.

ARCHITECTURE OF KMS

The decisive factor in selecting the optimal storage strategy for any enterprise application is the nature of the content and means to exploit its features. The need for a database for such a content/knowledge management system is governed by the usage of XML. It could be used as a data transport (data centric documents) or as documents (document centric documents) that serve as integral building blocks. The data centric documents are highly structured as opposed to unstructured varying length document centric documents. With XML as a data transport would mean that it would suffice to use a relational database and software to transfer the data between XML documents and the database. The use of integral XML documents, however, would entail a Native XML Database (NXD) that allows us to preserve physical document structure, support document level transactions and execute queries in an XML query language. This is possible due to the inherent "database" format features of XML in that it is self describing, portable (Unicode), and with description of data in terms of trees or graph structures [7]. The KMS (built on top of an NXD) is designed to use the key components of XML and its surrounding technologies, namely, storage, schemas, query languages and their programming interfaces.

Three-tier System Architecture

The present KMS implementation employs the three-tier system architecture to provide performance and scalability. In particular, the client tier is responsible for the presentation of data, receiving user events, and controlling the user interface. This is primarily made possible through the ASP.NET pages that execute on the server and generate markup such as HTML, WML or XML that is sent to the browser (Figure 1). The applications use ASP pages that apply an XSL style sheet to the XML documents retrieved from the document base in order to convert the documents to HTML and display them with the Web browser. With most Web browser supporting XSLT processing, the client side XSLT processing increases the application efficiency.

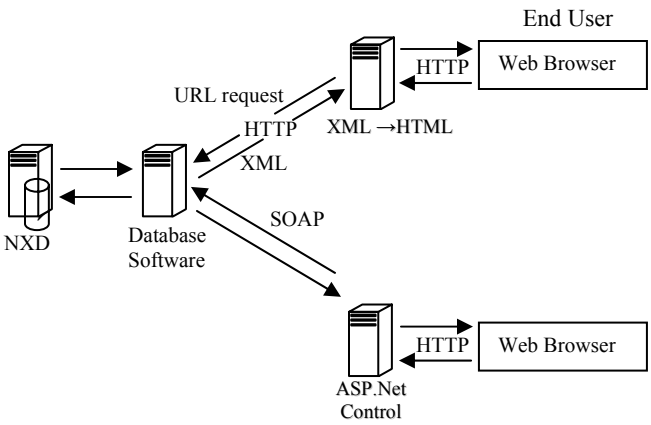

Figure 1 Data Flow Diagram

The client applications were developed programmatically using each of the API components and the API utility components. The API components provide methods and properties for building applications that communicate with the server. These components are dedicated to managing the server as opposed to the API utility components, which enable the server to manage a list of items it receives and returns. Each of these components provided a well defined interface to perform specific tasks such as providing access to the server and its administration services, document base installation, connecting to a specified document base, etc.

The second tier (also known as the Application-server tier) protects the data from direct access by the clients. The application layer consists of the logic and transaction capabilities between the user interface and the data layers [8]. The Server COM API is used for manipulating and querying XML. Web-based or stand-alone applications can be created that use the server's capabilities, enabling the client applications communicate with the server. The end users can access the document bases of this system through the Internet, intranet and the wireless networks using custom user interfaces. Other applications or computers can access them through the API. The Web server employed is the Microsoft IIS for Windows and

ASP is used as the Web application server to provide fast programming of interfaces and links to databases for the Web environment.

The main feature of this KMS prototype implementation is that the system is neatly structured, and that there is a well-planned definition of the software boundaries between the different tiers (Figure 2). The KMS architecture incorporates an NXD as its last (data-server) tier. The XML streams flow into the server where it is dynamically indexed and stored natively. The NXD is employed to store, index, and retrieve large volumes of XML and binary content. Unlike most XML repositories, the server works in a manner wherein it doesn't touch the native XML instance, rather it parses the XML instance and builds indices based on markup [3]. The result is a streamlined database structure, which gives rise to superior content indexing and content retrieval performance. Its design with XML at its core makes it highly versatile. The applications use the server as XML document management software. Some of the prominent features of this NXD are use of a smart proxy, background indexing, and thread management.

MAIN KMS FEATURES

The user can browse the contents of the KMS either via a "top-down" approach using the automatically updated table-of-contents (tree of documents) or through the "bottom-up" approach with the built-in search engine as shown in Figure 3. The information access and management functions are implemented programmatically in the KMS that allow authorized users to check-in and check-out documents. The document base has a document repository, managing the check-in, check-out, and versioning of XML documents that are held as self-contained entities in the database schema. Information can be securely checked in/out, classified, and organized with meta information provided by the users, making it possible to create an ever-increasing base of expertise. The checked-in documents are programmatically saved to the server. The check-out utility works in two different modes (normal and locked operation modes).

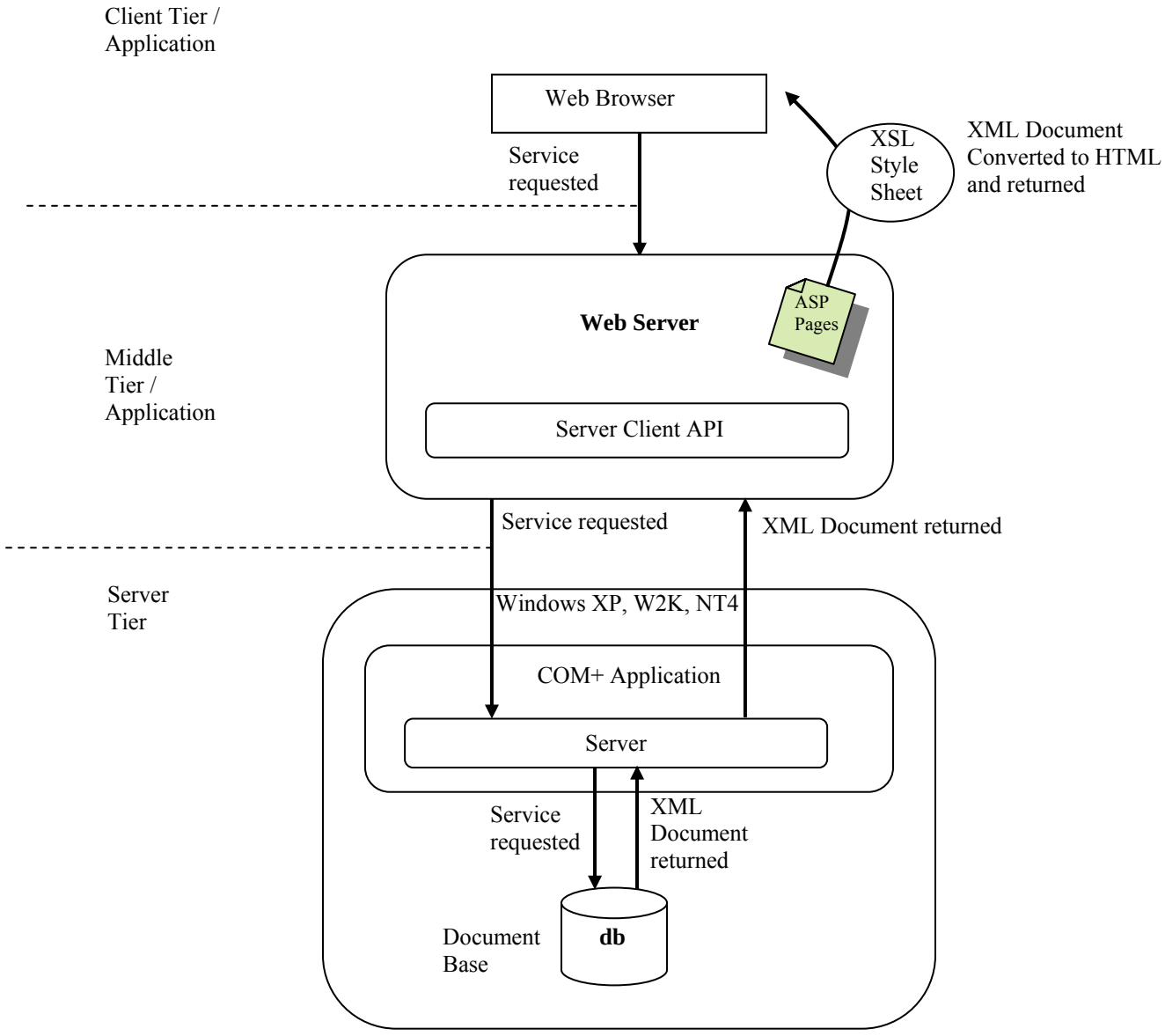

Figure 2 KMS's Three-tier Architecture.

The KMS provides an intuitive user interface wherein it manages the growth of user-developed information by providing the administrative and editing tools necessary for authorized users to generate customized knowledge on the fly for the XML Repository. Automatic indexing (using an Index Definition Document) that synchronizes index content with application processes is programmatically achieved which enables search on the XML knowledge repository, thereby,

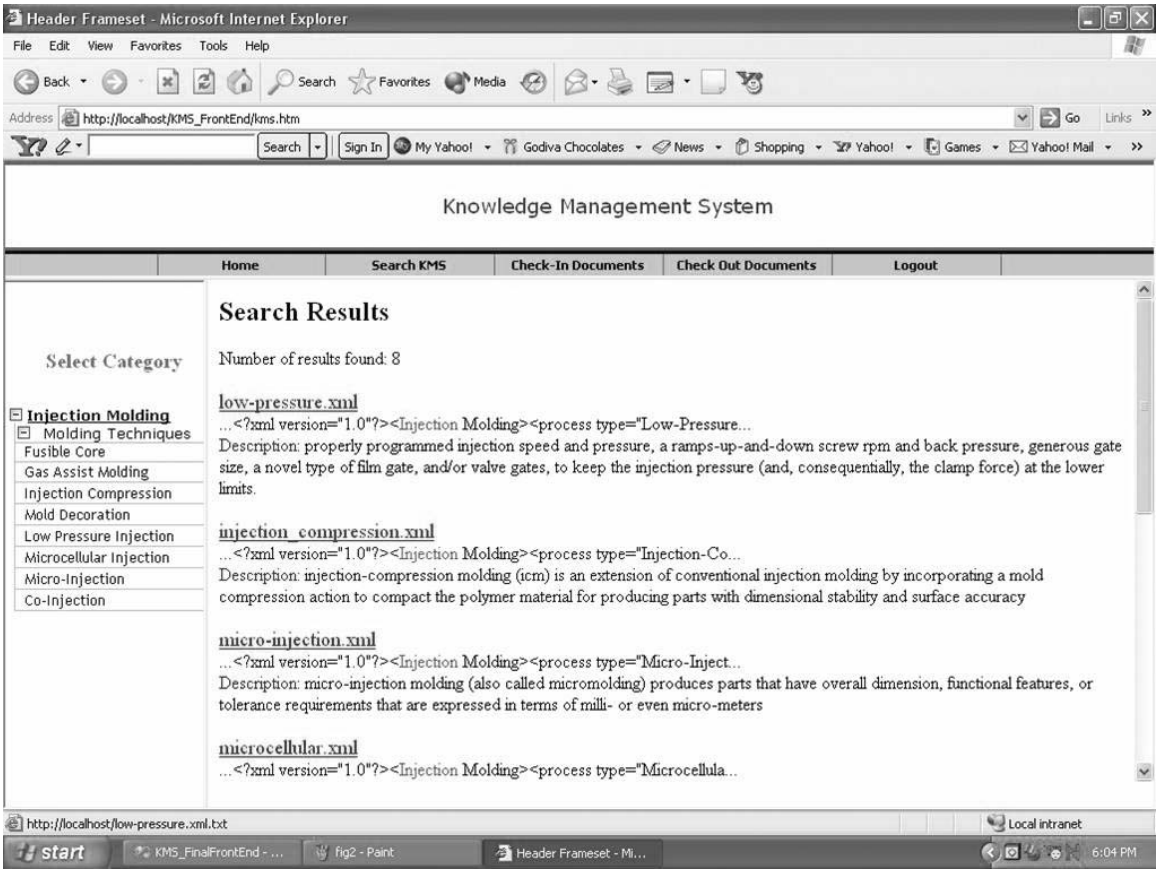

Figure 3 A Search Query yielding XML Documents

resulting in relevant documents in result spaces with abstracts and highlighted key words (cf. Figure 3). The dynamic indexing capabilities of TEXTML Server updates document indexes as documents are added to a document base, making it possible to search and retrieve documents immediately and optimize server resources. In addition to the search capability, navigational aids in the form of JavaScript driven hierarchical menu trees are provided to supplement the search with traditional hierarchical organization of topics. Queries to search the document bases are written in XML.

This KMS model includes arbitrary levels of nesting and complexity, as well as complete support for mixed content and semi-structured data. This model is automatically mapped by the NXD into the underlying storage mechanism. The mapping used will ensure that the XML specific model of the data is maintained. The KMS incorporates efficient indexing through the server's capability to logically group semantically yet syntactically different elements or attributes in the same index. It can index document properties and efficiently combine queries across indexes of multiple types. The indexing procedure can dynamically update indexes incrementally and automatically based on multiple criteria. With only specific information to be indexed, there is a decrease in the database overhead.

Administering the interactive document bases of this system enables the client applications to communicate with the server. The end users access these document bases using a custom interface and other applications enable access through the API. An XML utilities module checks the validity of the XML (using a validating reader) documents that need to be added to the document base. This module could also be used for primary XML document manipulation (as retrieved by any XPATH expression) or to apply effective XSL transformations (Figure 4) and generate the processed output to an output file.

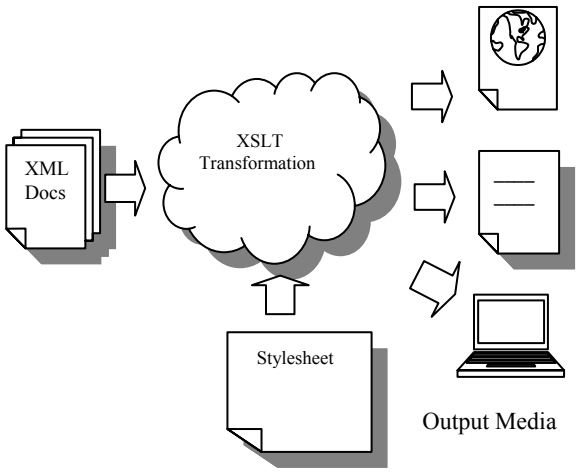

Figure 4 Transforming XML Documents

CLIENT API IMPLEMENTATION

The server is integrated to Microsoft Component Services (CS). It runs within CS environments and uses COM facilities of CS to perform all of its functions. An IIS is used as the Web server. The COM API includes both the server components (dedicated to the server) and components for any user application. The API components are used to develop applications that interact with the server. The server API hosts the server functions and the proxy (client API) hosts the service components that communicate with the server API. The components used in each of the client applications are classified as the server components, which are service components dedicated to the server. The other general components are not dedicated to the server. These components are effectively used to develop the client API to communicate with the server.

Based on the available interfaces each of the associations of the namespace prefix with a relevant URI reference is declared and specific libraries invoked. XML namespaces provide a simple method for qualifying element and attribute names used in XML documents by associating them with namespaces identified by URI references. The client application has been developed on the Microsoft .NET platform in C#. To enable ASPX debugging and compatibility, the "compilation debug" and "aspcompat" parameters are set to "true," respectively. Also the trace-enabled parameter is set to true to enable application trace logging.

Structure of the Document Base

The Document Base has a repository, which hosts all the documents as opposed to any structure of the documents itself. This is governed by the Index Definition document, a system document that contains the structure of the indexes created. This structure is dependent on the structure of the XML documents. The structure of the index could be defined using any of the optional reference lists, indexes, units, or descriptions. The indexes are associated with validation lists/stop lists, which serve as a guide/semantic delimiters for the user to perform keyword searches on the document bases. The stop lists are ignored during the indexing process. The indexes defined could be of various types such as the word, string, date, time, etc. Any index would need a dictionary of indexed terms and a list of occurrences specifying the relative position within each document. During the indexing process the server searches each XML element for these indexable values within the depth of the element relationships defined.

The structure of the indexes is analyzed for every search operation, which requires a check-out of the index definition document effectively as a system document. This also entails a persistFile object to attach the content and copy the relevant document to the local disk. Upon modification, the document could then be programmatically checked-in, which is a primary step for any indexing process. This is followed by an update of the document base using a DocBaseAdminServices object

Creation of Document Bases

This process is outlined in the process diagram plotted in Figure 5. This entails connecting to the target server with the new document base. With access to the server's administration services, the document base is installed. One needs to instantiate a ClientServices object that enables communication with the server, a PowerServices object that provides access to the server's application services, a ServerServices object that provides access to the server's services, and a ServerAdminServices object that provides access to the server's administration services. With the creation of every document base, one needs to define the structure of the indexes in the Index Definition document.

Check-In Utility

With connectivity to the server and the document base established, each of the user documents are checked in as shown in the process diagram (Figure 5). This requires creation of a dynamic list to add each of the documents to the document base. With a persistFile object (to retrieve the document's content) created and the contents of the file loaded into this object, each of the files is processed by moving the content to a Document object. This document object has the document's filename, Mime type and content assigned to it. Finally, this document object is added to the dynamic list created. When all the files are added to the document list, they are then added to the document base using the DocumentServices object's SetDocuments method. This check-in feature can be used for adding both standard XML documents and the system documents such as the Index Definition document. As documents are checked-in, each one is parsed and indexes are built as specified by the index definition document.

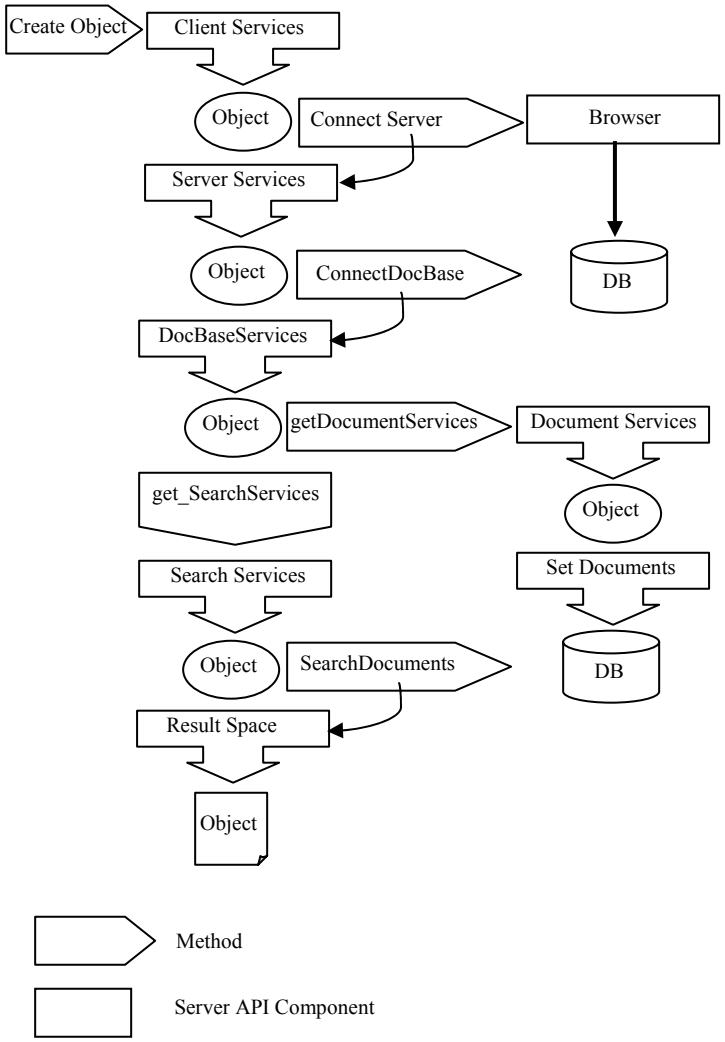

Figure 5 Objects instantiated in sequence using each of the Server API components. DocumentServices: provides access to the services for managing documents in the document base. SearchServices: provides access to the search services. ResultSpace: used to manage the results of the query

A file upload utility is used to prompt the user to check in documents to a specified destination. By ensuring that the Web server has write access to the file server, these documents are also stored on the file server. Since the KMS has a built-in hierarchical structure of documents (similar to the UNIX

directory structure) one could store the same document under multiple categories. This multi-category check-in is incorporated with the help of a repository, which is used to retrieve the category of each of the documents that needs to be either searched or checked in.

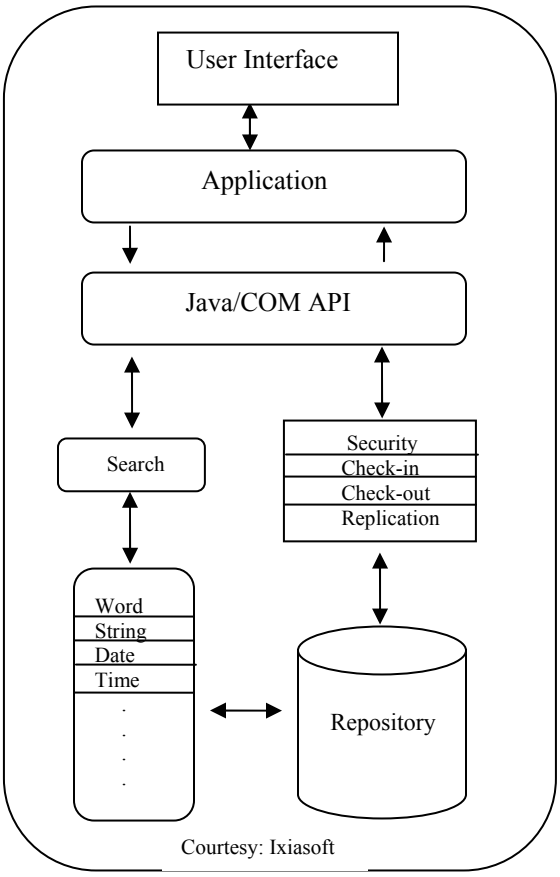

Figure 6 KMS Architecture

The ability to logically group related elements and attributes into indexes enables precise queries and optimized performance. A Java/COM API could be employed for the client application (Figure 6). This prototype incorporates use of this dynamic indexing capability.

One of the primary methods of this global repository class is to fetch the filenames sharing a specified category. Using the XML utilities module each of the file lists is stored in a predetermined location on the file server. This could be viewed as "virtual" categorizing of data since the server actually interfaces with a flat structured repository for all the user documents. The tree diagram for multi-category search/check-in is as shown in Figure 6.

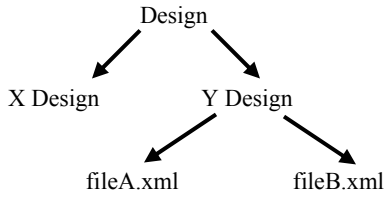

Figure 7 Tree Structure for Multi-Category Search Path for fileB.xml- (/Design/MoldDesign/fileB.xml)

Search Utility

The programming interface is shown in Figure 5. The search results obtained with an XML query are stored in each of the result spaces (logical locations) generated. The index queries search the content of the associated index as opposed to the document queries that search the document base resulting in retrieval of the documents using these references. Complex text and Meta data searches can be performed with complex queries on a single or multiple keys incorporating various operators, wildcards and combining result spaces. The SearchServices object's SearchDocuments method generates the search results for a given query. The result files are generated using a persistFile object as before and the XML utilities module used to parse it and store it in XML format. The module is also used to translate the result XML files using XSL transformations.

Checkout Utility

To edit any document, it must be requested from the server using the Server API, and have the server copy the document on a local drive. This would effectively mean that a file "check-out" is performed. A dynamic list to update the names of the documents to be checked out is created. Using the DocumentServices object's (provides access to the document services) GetDocuments method, the specified file chosen by the user is thereby called from the server. When this file is checked out, any user has only read access to the locked document. The checkout utility works in two different modes (normal and locked operation modes). In the latter mode, the document base refuses all modification operations but allows the user to view and search its content. This is a feature incorporated to control content editing.

APPLICATIONS

An organization's ability to learn faster than its competitors may be its own sustainable advantage. Such a prototype would enable network enabled transfer of skills and knowledge for Internet-based collaboration or Web-based learning (see, e.g., [9-12]). Its potential also lies in Distance learning with specified document security provided. This would entail restricted access to authorized users or instructors. In a collaborative project, each of the participants would have access to the other updated documents by their teammates. For e-learning applications, such a system could be used for transfer of data among various departments with categorized information about student records. These applications provide an ideal example for reusability with the knowledge framework for an organization mentioned earlier. This is similar in concept to a reusable Web service.

CONCLUSIONS AND FUTURE WORK

A KMS prototype has been designed and developed to provide a collaborative environment for any knowledge-intensive organization to create, capture, organize, and share knowledge. Through the use of an NXD server and the various documents administration features, it enables organizations seamlessly integrate KMS with their proprietary databases and help the organization to consolidate knowledge scattered in database records, electronic files, and papers. Future work on the KMS involves building a high level of intelligence into the search engine in terms of its ability to retrieve relevant search results (based on documents most viewed, with highest number of keyword occurrences, etc.) across categorized data. It is needed to ensure that the system be scalable wherein it can store, index, and retrieve millions of documents. Powerful multi-criterion sorting needs to be implemented allowing a customization of the order in which results are presented.

ACKNOWLEDGEMENTS

This research was supported by the Applied Research Program of the University of Wisconsin Systems. The authors are thankful to Ixiasoft for making the TEXTML Server available for the development of KMS.

REFERENCES

[1] Liebowitz, J. and Wilcox, L. C., ed., **Knowledge Management and Its Integrative Elements**, CRC Press LLC, 1997.

[2] Liebowitz, J., ed., **Knowledge Management Handbook**, CRC Press LLC, 1999.

[3] **The XML Content Server**, Ixiasoft Technical Paper, June 2002, www.ixiasoft.com

[4] Jon Bosak **Four Myths about XML**, Sun Microsystems IEEE Computer, (Vol. 31, No. 10, October 1998, pp. 120-22).

[5] Jon Bosak, **XML, Java, and the future of the Web**, Sun Microsystems, March 10, 1997.

[6] **XHTML™ 1.0 the Extensible HyperText Markup Language** (2nd Edition) A Reformulation of HTML 4 in XML 1.0 W3C Recommendation 26 January 2000.

[7] Diane Humetewa, Daniel Baker **Web Based Knowledge Management Systems**, GITA Conference 2001.

[8] Michael C Daconta, Leo J Obrst, Kevin T Smith, **The Semantic Web: A guide to the future of Web Services, XML and Knowledge Management**, Wiley Europe**.**

[9] Joseph M Firestone **Enterprise Information Portals and Knowledge Management**, Butterworth-Heinemann; 1st Edition, October 2002.

[10] Adam Freeman, Allen Jones **Microsoft .NET XML Web Services**, Microsoft Press, October 2002.

[11] Turng, L. S., et al., "Application of Internet and Web Technologies for Management of Molding Know-How," **Society of Plastics Engineers Tech. Papers**, 46, pp. 596-600, 2000.

[12] Turng, L. S. and DeAugistine, D., "A Web-based Knowledge Management System for the Injection Molding Process," **Plastics Engineering**, pp. 47-50, December 1999.