RJ10403 (A0701-015) January 25, 2007
Computer Science

IBM Research Report

Holistic Management of Integrated Content in Enterprise Information Systems

Mehmet Altinel, Kevin Beyer, Hamid Pirahesh, David Simmen
IBM Research Division
Almaden Research Center
650 Harry Road
San Jose, CA 95120-6099

Holistic Management of Integrated Content in Enterprise Information Systems

Mehmet Altinel Kevin Beyer Hamid Pirahesh David Simmen

IBM Almaden Research Center
650 Harry Road
San Jose, CA 95120
{maltinel,kbeyer,pirahesh,simmen}@us.ibm.com

ABSTRACT

Today's enterprise systems typically include both data-centric and document-centric applications. Data-centric applications are built on top of DBMS products which have excelled on advanced query processing and ACID transaction support for structured data. On the other hand, document-centric applications usually rely on content management system (CMS) products to perform advanced unstructured data management operations due to inherent differences in the usage patterns and required feature set (e.g. versioning, records management, etc.). We observe that a new class of hybrid applications are emerging that require the combined set of DBMS and CMS features on structured and unstructured integrated content due in large part to increasingly complex business requirements and the widespread adoption of XML technologies. However, today's hybrid applications are forced to fragment their business artifacts in separate DBMS and CMS repositories, and cope with accessing, augmenting, and processing the separate pieces. The lack of a unified repository model for integrated content makes the development of hybrid enterprise applications painfully difficult, and often leads to short-lived, inadequate solutions. In this paper, we explore the trends in hybrid enterprise applications and their requirements for a unified repository model. We suggest a holistic approach for the design of the new repository model covering both DBMS and CMS features under one umbrella. We discuss the integration challenges, and present our experience with a prototype that we developed in the MUSIC (Management of Unstructured and Structured Integrated Content) project.

1. INTRODUCTION

Over the years, the database research community significantly contributed to the spectacular success of database management system (DBMS) products. These products have excelled in core structured data management features such as advanced query processing, high throughput transactional store, highly optimized concurrency control, and data recovery. Today, relational DBMS products enjoy a $15 billion market (according to IDC [40]). Furthermore, there is a sizable DBMS eco-system built around the core features with many other new technologies such as ETL tools for data collection, data warehouses, OLAP tools and data mining solutions. All these products and technologies gave rise to widespread deployment of data-centric applications in enterprise systems.

On the flip side, an overwhelming proportion of information available on the Internet and corporate intranets still resides in unstructured form (usually as a document), out of reach of DBMS products. Document-centric applications have been developed to render many different forms of unstructured data such as reports, images, spreadsheets, emails, blogs, forms etc. Commonly used Web portals, office products, and source control systems (e.g. CVS, ClearCase) are well-known examples of document-centric applications. Since the document usage pattern widely differs from structured data, document-centric applications need a different set of features such as versioning, check-in/check-out, long duration locks, advanced search, policy-based retention and access control, which are not delivered by typical DBMS products.

Content management systems (CMSs) have been introduced to provide the required support for the above features (e.g., [12][13][15]). Document-centric applications utilize CMS products for creation, management, distribution, publishing, and discovery of a variety of content (such as text, graphics, video, documents, etc.), and providing connections to corporate workflows.

Enterprise system infrastructures today have become overwhelmingly complex with unprecedented number of data-centric and document-centric application silos. Historically, these applications are characterized by the functional boundaries mostly defined with the capabilities of underlying DBMS and CMS products. However, as the information systems are getting more complex, we see more cases today where the applications are forced to push the boundaries by managing structured and unstructured integrated content together. Developers that use only one of two systems are forced to implement missing management features of the other system in their application. For example, some wiki systems built on a DBMS implement versioning themselves. Such additions usually provide only a limited functionality, which are "good enough" for simple applications. However, most hybrid enterprise applications cannot afford to implement the required features of the "other" system. As a result, an increasing number of enterprise applications utilize both a DBMS and a CMS repository to manage their hybrid business artifacts, and are forced to deal with the integration problems. For example, in a typical healthcare management system, personal information of a patient and his/her financial transactions could be handled with the help of a DBMS. On the other hand, a patient record must also include other forms of data such as doctor visit reports, test results such as X-Rays, MRI images, etc., which need to be stored in a CMS for easy management. Another example is the insurance industry, which relies on structured data, for tracking customer payments and performing risk analysis, as well as documents, such as policies, claims forms, photos of damage, and accident reports for insurance records.

We anticipate that such hybrid applications are fast becoming a norm in enterprise systems as structured and unstructured integrated content turns out to be a natural artifact in a wide range of application domains. This new class of applications need to process integrated content by combining the technologies shown in Figure 1. They need advanced querying and optimization, not only to retrieve but to be able to perform complex composition, and transformation and most importantly advanced business analytics tasks over the integrated content. At the same time, they need higher level management functionalities like mixed transaction models, policy-based access and retention control, versioning, etc., which are commonly applied to the integrated content. Note that information integration technologies, in particular widely studied DB and information retrieval (IR) integration, can provide a relief for querying and optimization, but they fall short for other higher level management functionalities which regularly interact with the query mechanisms.

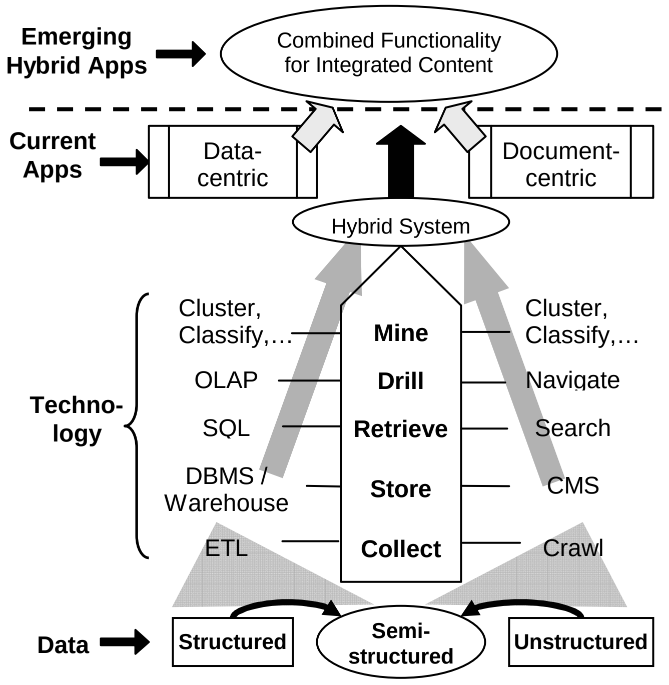

Figure 1: A view on convergence of DBMS and CMS

Our prediction on the accelerated adoption of enterprise hybrid applications is mainly based on the following observations:

Business Factors: Advances in digital technologies and communications gave rise to ubiquitous use of business artifacts with greater wealth of "content" than ever, in a multitude of formats such as images, text documents, Web pages, spreadsheets, presentations, graphics, drawings, e-mail, video and multimedia. Organizations are facing the increasingly costly challenge of managing this abundant volume of content. On top of that, there are several influential business factors that require coordinated access and higher level management functionalities for integrated content. Perhaps the most remarkable one is the Sarbanes-Oxley Act which The United States Congress enacted in June 2002 in the wake of several highly-publicized corporate scandals [3]. The act instituted a series of corporate reforms to improve the accuracy and reliability of financial reporting. Most notably, the act requires companies to implement systems of internal control over financial reporting. A reliable implementation requires new solutions to apply compliance rules for *all* kinds of data across information sources. Enforceable document retention policies (i.e. policy-based access and retention control), systems for managing integrated content, and consistent records management are all key requirements from a legislative standpoint.

XML Effect: The recent introduction of the XML standard gave rise to an explosive growth of complex XML representations with integrated content: (1) XML has been extensively used to create new business artifacts by combining different pieces of structured and unstructured data with information integration techniques [9][22]; (2) There are new XML-based standards that naturally define integrated content representations used in hybrid applications. Most notable examples include the following:

- In healthcare, the emerging HL7 Clinical Document Architecture (CDA) [21] standard is used for storage, distribution, and display of patient records.
- In finance and insurance, XBRL [39] has been proposed as a standard reporting language that captures the information rendered into a business report including fact data, basic calculations, presentation layout, etc.
- For forms processing, the XForms [37] standard provides a common representation for capturing and processing the documents in Web-based business workflows. It formats a document in several decoupled sections for data, logic and presentation.

CDA is already in use in many healthcare enterprises including Mayo clinic, Kaiser, Duke, etc [1]. Both XBRL and XForms are gaining widespread support from ISVs and leading enterprise application infrastructure vendors [21][39]. What is common in these standards is that they eliminate inherent separation between data and document, paving the way to use the same object representation (business artifacts) throughout all the enterprise system applications.

Furthermore, major office product vendors have started to offer new document formats in XML [1][25]. Documents created in this way contain structured and unstructured integrated content. A wide range of new applications and collaboration systems, most notably Adobe Intelligent Document Platform [1], and upcoming release of Microsoft SharePoint Server 2007 [28], aim to take advantage of this new representation.

Where are we at today?

It is evident that neither DBMS nor CMS technologies alone can be an answer to hybrid applications' demands today. Hybrid application developers today are dealing with this challenge by imitating missing functionalities inside their applications. For small scale solutions, they usually develop custom methods for missing features in a limited way over either a DBMS or CMS product. For example, it is very common to see a primitive versioning system developed inside the data-centric applications. At the enterprise level, however, they frequently need more sophisticated and scalable solutions. Thus, the developers often end up using both DBMS and CMS products through a complex integration layer implemented either inside or outside of their application. But in this case, they pay the price of dealing with the burden of fragmenting and storing the business artifacts in separate DBMS and CMS repositories, and then coping with accessing, augmenting, and processing the pieces. For instance, consider the patient record example given before. When the healthcare applications need to access a patient record, they have to deal with multiple data models (with different semantics), multiple APIs, and more importantly, optimizing access to multiple repositories. It is painfully difficult to generate reports and/or to perform business analytics over fragmented patient records. What's more, in many cases, replicated parts in multiple repositories could easily lead to consistency problems.

Highly restrictive compliance requirements are also becoming a considerable cost factor for hybrid applications. The lack of a single repository model for integrated content imposes a significant financial burden on the companies as enforcement and monitoring of internal controls consume numerous employee hours. Surveys of large public companies indicate that Sarbanes-Oxley compliance costs exceeded $4 million per company in the fiscal year 2004 [20].

What is missing?

We observe that DBMS and CMS technologies are fast coming together as the above factors force them to offer increasingly overlapping sets of features. They are trying to capture the emerging market of hybrid applications by attacking the problems from their perspective. However, the solutions created this way are often feature rich for one set of problems, but poor for the other. Hybrid applications need solutions that will work equally well for the entire feature set. As achieving this goal by looking at the problem from one technology side is not feasible, we suggest a new perspective, *a holistic approach*, for the management of integrated content, covering both DBMS and CMS features under one umbrella. We try to find an answer to the growing demand by exploring a unified repository model. We believe that the convergence of DBMS and CMS technologies (and IR, as an integral component) should be analyzed from the applications perspective rather than the technology perspective. We are working on the MUSIC project (Management of Unstructured and Structured Integrated Content) to fulfill this goal.

1.1 Contributions of the Paper

Main goal of this paper is to investigate the convergence of structured and unstructured data management from an enterprise application's point of view. Our investigation approach is distinguished from the previous work on database and IR integration, and information integration in general, in two ways: (1) Our starting point is the natural and widespread existence of complex integrated content, especially in XML form, as a base artifact in enterprise systems; (2) Analysis of the problem is not solely limited to a querying perspective. Instead, we take a broader view by considering all aspects of information management covering both DBMS and CMS features. We believe such a broader view is necessary since all the elements in the unified feature set interact with each other.

We try to make a case that there is a substantial value to develop a new repository model unifying data-centric and document-centric features for integrated content. And, XML technologies can provide the needed foundation to achieve this goal. Our contributions can be summarized as follows:

- We examine the existing technologies for DBMSs and CMSs, and identify how these technologies are related to each other.
- We provide a detailed discussion on why the convergence of DBMS and CMS technologies is inevitable, and elaborate on the challenges and open issues for database researchers.
- We present our experiences with a MUSIC prototype that loosely integrates relational DBMS and CMS products. The prototype demonstrates a primitive example of unified repository model developed by leveraging salient features of DBMS and CMS technologies in a complementary way. We use the prototype to give some hints about what such repository can offer for hybrid applications.

It is not our intention to propose a specific architecture or method for the new repository model. We rather advocate a holistic approach for the management of integrated content by identifying the requirements and research challenges from the perspective of emerging enterprise applications.

1.2 Organization

The rest of the paper is organized as follows. We first provide background information on CMS technologies in Section 2, and examine how DBMS and CMS (and IR) technologies are used together today. Then, in Section 3 we identify the trends in these technologies and present our predictions for future information management systems. We also elaborate on research challenges and open issues in a unified repository model. Section 4 reports our experiences with an early repository prototype that we built to support hybrid applications. Section 5 presents related work including a short literature and industry survey on DBMS, CMS and IR integration. Finally, we provide concluding remarks in Section 6.

2. A SNAPHOT ON CMS TECHNOLOGIES

A typical enterprise CMS contain components for high level semantic functions like document management, collaboration, business process management, records management, email management, workflow and Web content management. From the data management perspective, enterprise CMSs are designed to provide high volume storage capacity, and fast key-based access for document retrieval. In general, they offer a procedural API, which is coupled into high level programming languages. Unlike DBMSs, their query support is limited to document retrieval only and it lacks advanced query features including joins, aggregations and composition. These operations are usually handled by CMS applications. Naturally, CMSs utilize IR techniques, and sometimes external IR systems, to retrieve the documents.

Compared to the DBMS market, the CMS market currently has a smaller size[1], but this market is very dynamic and it is growing at a faster pace. This market is highly fragmented with many different types of open-source and proprietary content management solutions. This fragmentation can be rooted to the definition of content. Every piece of information that is stored digitally within an organization can be described as content, and

[1] Market size estimates range anywhere between 400M to several billion US$ depending on how the boundaries and functionalities of a CMS is defined.

every CMS vendor sees the management of content from their product perspective. For the sake of simplicity, our focus in this paper will be on common document management features including policy-based access and retention control, text and metadata search, content and metadata creation, versioning, multi-user authoring, publishing, and connecting documents into workflows. Section 4.1 delves into details for these features, through describing a recently proposed API standard, called the Java Content Repository (JCR). More detailed description of enterprise CMS features can be found in [11][24][28][32] and numerous Web sites on the Internet.

2.1 Role of Metadata and DBMSs

Documents are fundamental entities in a CMS in the sense that they are managed as a unit. Actual document content is opaque to the CMS. For that reason, they expect the users to extract and attach structured or semi-structured metadata information to the documents to enable faster and more accurate search. Otherwise, key based access methods (and/or search mechanisms for text based-documents) are usually the only options for document retrieval. The metadata is also useful for categorizing and linking documents together.

Most CMSs utilize an embedded DBMS underneath to store application specific metadata as well as system metadata. The CMS exploits system metadata such as timestamp, item size for its content management functions. Applications can create their own metadata for fast document retrieval. Moreover, DBMSs' strong transactional storage and recovery features are used to keep the system state intact. As for the actual storage of documents, CMSs usually prefer file systems or high-end storage systems. Thus, a typical CMS orchestrates both database and file system management tasks. Occasionally, they may inline the storage of small size documents with the metadata in a DBMS for better performance.

There are several challenges for utilizing DBMSs. Perhaps the most important one is the mismatch between data models. In general, CMS data models support nested type hierarchies to represent high-level content definitions consisting of metadata and documents. This mismatch creates major problems in mapping CMS types into flat database tables. Some DBMSs provide object-oriented features to help with this mapping. But the real problem is that specified types in CMS applications are very dynamic and structures may widely vary. Hence, even a moderately complex CMS application may end up generating lots of different content types mapped into overwhelmingly high number of tables in the underlying DBMS. The queries issued to the DBMS rapidly become very complex[2] and hard to optimize.

Recently available XML support in DBMSs can provide some relief in metadata management since the XML data model is more natural fit for CMS due to its flexibility. Some CMS systems (e.g. [13][21]) already provides XML mapping of the repositories for easy content import/export. XML support of the DBMS, however, should not impose strict schema conformance rules on the metadata representation.

[2] Due to the dynamic and hierarchic characteristics of CMS data models, even a simple CMS query may turn into a large and complicated recursive SQL statement. Understanding such a statement is a challenge even for an experienced DBA.

DBMSs can be improved to provide a better support for CMSs. DBMSs lack scalable large object (LOB) support, which is fundamental in CMSs. Moreover, a CMS must handle asynchronous delivery/loading operations to accommodate content streaming, and to sustain acceptable throughput rate in document ingestion for a high volume production system.

There is also wide mismatch between a DBMS and CMS security models. The latter needs to support more sophisticated, policy-based access control checks. DBMS's event generation and workflow support are usually insufficient for CMSs. Finally, policy-based retention control feature does not exist in a DBMS.

CMSs usually hide the existence of their underlying DBMS from their applications. In such an embedded setting, autonomic management features of the DBMS become very critical. This is important not only for lower total cost of ownership (TCO) but also for sustaining acceptable performance for highly dynamic CMS settings.

2.2 Interaction with IR Technologies

IR techniques play an increasingly important role in CMSs. The sheer volume of unstructured data easily makes metadata generation and maintenance a very complex and overwhelming task for the users. As a result, metadata structures usually become incomplete and unsynchronized with dynamically changing content. It is much harder to locate the documents with such poorly specified metadata. IR technologies can provide a relief to cope with such dysfunctional metadata.

Integrated text search is already a common feature in CMSs [12][13][15]. Usually an external text index system is utilized to enable this functionality. Major DBMS products today also utilize a text index and provide integrated full-text search feature. Some CMS implementations take advantage of this feature instead of using an external text index.

Recent advances in semantic search are leading to more effective document retrieval by automatically generating additional metadata [34]. Automating the metadata generation creates a new set of challenges in terms of management and exploitation. Although this technology is still in its infancy, it has the potential to become an indispensable component of CMSs.

3. AN ANALYSIS ON DBMS + CMS + IR CONVERGENCE

Structured data management is now considered a mature field. Over the years, the database community has proposed several extensions to structured data models like object-oriented data models and query languages. Recently, there has been a growing interest on bringing together DBMS and IR systems mostly from the perspective of integrating the two different query paradigms. Parallel to these efforts, CMSs, have been enhanced for better exploitation of IR technologies. In this paper, we investigate the convergence at a broader level, including all the three systems together. Figure 2 illustrates our perspective on the converging trends from the diverse areas of the information management system.

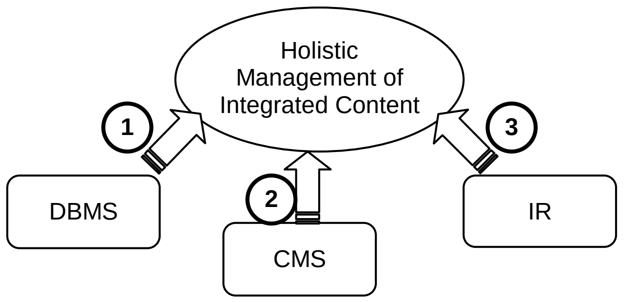

Figure 2: Convergence of DBMS + CMS + IR Technologies

(1) DBMS products have been trying to "move up in the food chain" by subsuming more high-level semantics inside their engines. Today, all major DBMS vendors (IBM DB2, Microsoft SQLServer, Oracle DB) have announced engine level support for XML data management. In particular, Oracle 10g (release 2) took a more aggressive step by introducing basic content management features (WebDAV support, foldering operations, etc.) and querying functionality for XML data [31]. Furthermore, the database engines offer ranked text search on XML and other text documents.

(2) CMSs have long being touted as being managing structured and unstructured data. Over the last decade, the CMSs have gradually evolved from vertical, specialized, line-of-business applications to horizontal system infrastructure that is targeted to support wide variety of content applications. The latest and most serious step in this direction is the proposal of a standard API, called the Java Content Repository (JCR), as described in Section 4.1. The JCR standard provides a semantic data model, and a query language for capturing management aspects of both structured and unstructured data. Most importantly, the JCR has strong support from CMS industry leaders, including SAP AG, Macromedia, and IBM, establishing its use and importance in the enterprise landscape. We anticipate that the standard will continue to grow to provide more elaborate features for structured and unstructured data.

(3) Explosive growth of unstructured data available on the Internet and enterprise systems have made text search engines a critical part of a wide range of applications. To further improve the precision of the results, research efforts for text search has started to concentrate on using more semantic information in ranking algorithms. Semantic information is distilled from unstructured data and it reveals itself in some form of structure (e.g. RDF). Semantic search is currently a very active research area in the IR field [38], and new technologies have recently started to emerge. For example, IBM's UIMA (Unstructured Information Management Architecture) platform attempts to provide an extensible tool to develop customized semantic analysis and search components for unstructured data [34].

Besides improving the precision in search results, semantic information can be used to analyze massive amounts of unstructured and semi-structured text for the discovery of trends, patterns and relationships from data. The WebFountain project at IBM Research [35] aims to realize this vision by attaching structured information as annotations into the unstructured text documents.

3.1 Design Space and Research Challenges

The new repository model must be designed by bringing together DBMS, CMS and IR technologies. The combined feature set should be explored to understand the interactions between its elements. IR technologies have been independently employed by both DBMS and CMS products for a long time. But, when the three technologies come together, additional requirements must be taken into account: Does the text index need to be aware of access control or document versions? Should table records be versionable? Should all large objects in the database be versionable? What new query features are required for content? What is the right API to the combined system?

At this point, it is unclear what the combined architecture should be. Moreover, different architectures may be appropriate for different application classes. In Section 4, we describe our first prototype of a combined system, but we are far from completing our investigation. Much additional research is required to properly explore the space. In this section, we describe the design space and the key tradeoffs that must be made while building a combined system.

Holistic vs. Federation: A combined system can be built by a federation of a DBMS, a CMS, and an IR system, as proposed in [32]. Structured data goes in the DBMS, unstructured data goes in the CMS, and queries are federated across the two systems. Federation has the advantages that if the data already resides in the two systems, then it quickly brings combined query support without the need to migrate the data or modify the existing systems. Federation also grants the user the freedom to choose the CMS and DBMS best suited for their application. However, federation has its problems: Application developers and administrators must understand the quirks of two systems. This increased complexity significantly increases the total cost of ownership (TCO). Furthermore, the separation between the two systems greatly degrades the overall performance. We believe a holistic approach is preferred, which brings the data into a single repository to reduce the TCO and improve performance. The holistic approach also offers the chance to unify other aspects of the system like the data model, query language, transaction model, etc., as described below. Moreover, federation is inappropriate when a single business artifact is both data (requiring DBMS support) and content (requiring CMS support), which is the case for many hybrid applications.

Data Model: The combined system must bring tabular data, documents, and metadata together into a single data model. The system could model document collections as tables, or it could model tables as collections of small documents. A similar problem was faced when XML and XQuery were added to SQL [8]; should XML data be modeled as table, or tabular data be modeled XML? However, the border between the two worlds is a source of significant complexity and potential performance problems when the two models are out of sync. Is there a simple, unified model?

Concretely, applications should be able to store and query complex (XML) objects with integrated content, such as HL7 CDA, XBRL, or XForms instances, along with additional system and user metadata about the instances. The system must also have strong support for links between the objects.

Versioning: The system must provide support for versioning of content. To capture the concurrent creation of content, support for branching and merging of versions is required. However, many hybrid applications are satisfied with a simple linear version history.

Transaction model: Both regular ACID transaction models and long duration transactions for disconnected clients must be supported. Furthermore, the system should support concurrent, disconnected modifications by providing and efficient conflict detection mechanism. The concurrency models and recovery models should exploit the versioning.

Access control and retention: Content systems require a more sophisticated access control mechanism than database systems typically provide. Moreover, the new compliance regulations place additional demands on access control and document retention. How can we unify the authorization models of DBMSs and CMSs? Can the DBMS adopt the CMS authorization model without adding extra complexity?

We believe that declarative policy-based access control and retention mechanisms are required to ensure compliance and improve the auditing of the system. In other words, the access and retention of document is derived by a query that looks inside the data and metadata of the content to determine the document's status based upon its current state. For example, items cannot be added to a signed purchase order, and cancelled purchase orders should be deleted in 90 days, but fulfilled orders must be kept for 3 years.

Workflow Connections: The system must participate in larger workflows. It must generate events for state changes in the repository, manage metadata created during the workflow, and manage the access to the objects as its state changes during the workflow.

Metadata Management: There must be mechanisms to manage application specified and system generated metadata. Furthermore, the repository should support auto metadata generation on the content by using IR techniques.

Query Language: The system must provide declarative query processing over the content, which includes support for reporting (e.g., aggregation) and creation of new documents from the stored content (e.g., joins and transformations). (This is what DBMSs are very good at, and CMSs are typically lacking.) The system should be able to easily combine operational data with content data. The query language must have support for links between objects and support the schema flexibility inherent in content. The query language must include both strict predicates and relaxed searched conditions with ranking. The language must understand document versions and support historical queries.

The XML-based query languages, SQL/XML and XQuery, could be used with suitable extensions. An SQL interface is important for connecting to the large number of SQL reporting tools already available. However, the arms-length integration of XQuery into SQL is source of complexity that might be eliminated by a new query language. Is there a better way to integrate these languages? Is an improved XQuery the answer (e.g., XQuery extended for analytics and search)? Is there a better language?

Search Language: The system should also support access through a simple search language. The search language should be a part of the full query language above (what is the best way?), but the system should support simple search queries without the complexities of the full search language. It is likely that the search feature may be conversational (the user helps guide the search), support faceted search, or exploit semantic search. The search should include not only the text, but also the structured data and the links as well.

Modification Language: Like SQL, the system should have a language for inserting, deleting, and updating the content. In addition, the system must include support for checking documents in and out of the system, restoring versions, defining triggers to connect the content to workflows, etc.

Componentization: Both the DBMS and CMS architectures have to be thoroughly reevaluated to find reusable components across the systems. As a starting point, we used the query engine component of the DBMS for the CMS. Conversely, the CMS repository functioned as a document store for the DBMS. We must detail this functional separation to lower levels by more aggressive, deeper component identification and interaction strategies in both system architectures.

Performance: As always, all of the above issues have to be studied by taking into account the performance aspects of the integrated system.

4. A UNIFIED REPOSITORY PROTOTYPE

In this section, we present an early MUSIC (Management of Unstructured and Structured Integrated Content) repository prototype that we developed to investigate the main issues in the integration of DBMS+CMS+IR systems. By no means do we claim that this prototype is the best approach. It only presents an investigation of one alternative in the vast design space.

We paid more attention on the repository APIs than attacking the architectural problems. Our aim was to get an initial feedback about what a unified repository model looks like, and to build a sandbox for our future work.

In the prototype implementation, we used a relational DBMS (IBM DB2 UDB v9.1 Beta [8][14]), and an embedded CMS supporting JCR (Java Content Repository) API (DB2 Content Manager-Internal JCR [13]). JCR is a recently introduced standard to define a common programming interface for CMSs. We start the section by giving an overview of the JCR API. Then, after highlighting the main prototype components, we present the MUSIC repository APIs, and show how they can be exploited by hybrid applications. Finally, we touch several architectural issues in the prototype and provide our initial optimization techniques to overcome performance penalties for XML documents.

4.1 Java Content Repository (JCR) API

As the number of vendors offering proprietary CMS repositories has increased, the need for a common programmatic interface to these repositories has become apparent. Java Content Repository (JCR, also known as JSR-170) API standard was recently introduced to provide such an interface. Just as SQL transformed database programming, the JCR API is expected to change the way to design and develop CMS applications. In this section, we highlight main aspects of the API. Details can be found in [16].

Data Model:

A JCR content repository consists of one or more *workspaces*, each of which contains a tree of *items*. An item is either a *node* or a *property*. Each node may have zero or more child nodes and zero or more child properties. There is a single root node per workspace, which has no parent. All other nodes have one parent. Properties have one parent (a node) and cannot have children; they are the leaves of the tree. All of the actual content in the repository is stored within the values of the properties. Figure 3 shows an UML diagram of JCR data model.

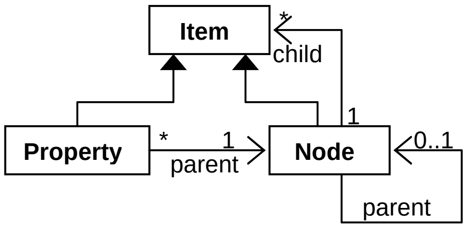

Figure 3: Elements of the JCR Data Model

Basically, the JCR API provides a set of Java interface classes and associated methods to perform content operations. Because nodes and properties have some common functionality, common methods are defined in the interface **Item**, to which the sub-interfaces **Node** and **Property** add further methods.

The JCR API was designed to support both hierarchical and non-hierarchical repository models. This is done by providing for both hierarchical, path-based addressing of content items, and direct, UUID-based addressing. Figure 4 depicts an example JCR representation of managed content for a real-estate Web portal application. In this example, the listing node contains two properties, namely description and mlsnumber, and a child node named reports, which, in turn, has two properties, inspection and picture. The description, inspection and picture properties have binary type to store XML, PDF and JPG documents respectively. The mlsnumber property is a string that stores the identifier for the listing.

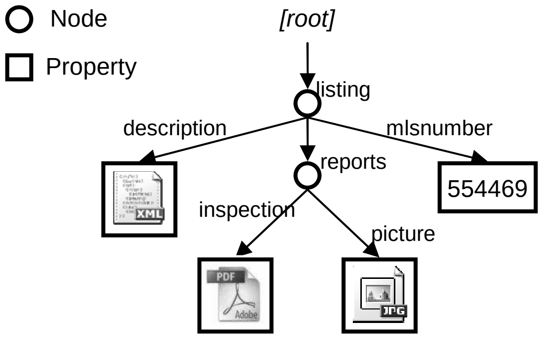

Figure 4: A JCR example for real-estate data

The JCR specification defines two XML mappings (*system view* and the *document view*) of the content repository data model. The entire workspace or subtrees within a workspace can be imported or exported with these XML mappings.

Type System:

JCR supports a very flexible and extensible type system. Every node *must* have one and only one *primary node type*. The primary node type defines the names, types and other characteristics of the properties and child nodes are allowed or required to have.

In addition to its primary node type, a node *may* also have one or more *mixin types*. These are node type definitions that can mandate extra characteristics (i.e., more child nodes, properties and their respective names and types) for a particular node in addition to those enforced by its primary node type. Similar to object-oriented models, node types can be defined in a hierarchy.

Query:

The JCR spec introduces two different query syntaxes: SQL-based (support is optional) and XPath-based. For the latter, JCR applications can search and navigate a workspace over the XML views. For example, for the tree given in Figure 4, the following query returns inspection reports for listing with mlsnumber 554469 in the document view:

/listing[@mlsnumber=”554469”]/reports/@inspection.

The result set of a query constitutes all the nodes in the workspace that meet the constraints stated in the query. The constraints fall into three categories:

- *Type constraint*: This limits the returned nodes to a particular primary node type (and possibly, additionally limits the nodes to those with particular mixin node types).
- *Property constraint*: This limits the returned nodes to those with particular properties having particular values.
- *Path constraint*: This limits the returned nodes to those within certain subtrees in the workspace.

The API also includes functions to embed a statement in a full-text search language.

Metadata:

An important aspect of the API is that it does not distinguish between “real” content and meta-content. The designers of the API believed that such a separation would only duplicate the entire API, since one would probably want to provide the same functionality for handling both meta-content and primary content. The distinction is in any case only meaningful at the level of the application, not the repository. Any particular application built on top of a compliant repository may, of course, choose which content is to be considered “meta”, and which primary.

Versioning:

In a repository that supports versioning, a workspace may contain both versionable and nonversionable nodes. If a node is versionable, its state can be saved for possible future recovery. Versions exist as part of a *version history*. Within a version history, the versions form a *version graph* that describes the predecessor/successor relations among versions of a particular versionable node.

Version histories and their contained versions are stored in version storage. The JCR API specifies interfaces for versioning operations including check-in/check-out, restore, update, and merge.

Locking and Transactions:

In JCR, locking allows a user to temporarily lock nodes in order to prevent other users from changing them. A lock can be specified as either *shallow* or *deep*. When a lock is placed on a node, it can be specified to be either a short-term (session-scoped) lock or a long duration (open-scoped) lock. As for the transactions, the JCR API relies on the Java Transaction API (JTA) specification [25].

Observation:

This feature enables applications to register interest in events that describe changes to a workspace, and then monitor and respond to those events. The observation mechanism dispatches events when a *persistent change* is made to the workspace. By the way of observation events, a JCR repository can be connected to a specific workflow.

4.2 Prototype Components

The DB2 UDB version used in the prototype provides support for storing, managing, and searching XML data. In the prototype, the JCR system is configured to use a DB2 database as its data source. Currently, this JCR system is used internally in the IBM WebSphere Portal product [36] and it is not available as a stand-alone system.

The MUSIC repository provides both a relational DBMS (SQL) and JCR view to its applications. Internally, we leverage complementary features of the two systems: DB2 is used for advanced querying, and JCR is used for document management tasks. We configured the JCR system in a way that it uses the application database instances instead of creating its own database. Finally, we created a special mixin type for the JCR repository to store required information for the MUSIC operations[3].

For the API design, we took a functional approach. We defined two sets of functions (1) to query (retrieve and extract), (2) to perform document management operations. With the help of these functions, the MUSIC repository can serve hybrid applications. Of course, existing data-centric and document-centric applications can also benefit from the extensions.

We decided to use the declarative languages, SQL and XQuery as the language bindings for our functions. The functional approach allowed us the needed flexibility in the language binding. We chose SQL as the main binding language for enabling full functionality, while allowing only the querying functions for XQuery. We could have used a high-level programming language such as Java for the binding. We did not take this route because (1) SQL is a highly powerful language, offering elaborate mechanisms for data manipulation, (2) declarative nature of SQL allows optimizations for the execution, and (3) there are many existing tools and applications providing sophisticated reporting and OLAP tools using ODBC/JDBC and SQL. With a SQL binding, we take advantage of all these powerful features for the documents in the MUSIC repository. And, by providing XQuery binding, we facilitate access to XML documents stored in the MUSIC repository for emerging XQuery applications.

As our initial focus is on providing the right functionality rather than performance, our methods are intentionally designed not to be invasive (i.e. only minor modifications in DB2 and JCR engines). The MUSIC functions are mostly implemented using SQL and JCR APIs. We also exploited DB2's information integration support to glue the systems. The combined system is good enough to offer the required set of features for both structured and unstructured data management. In the prototype, our primary focus is particularly on the management of XML documents[4].

Figure 5 illustrates the high-level design and alternative uses of the prototype. Hybrid applications can use SQL views with MUSIC functions for managing integrated content. It is also possible to perform document management tasks through the JCR view. In SQL view, documents are conceptually inserted and stored in table columns. On the other hand, in JCR view, they are part of a JCR tree. Regardless of which view is selected, hybrid applications can insert and work on the same document collections with the MUSIC functions. We will elaborate more on these views in the next section. Existing applications can leverage the MUSIC functions as well. For example, document-centric applications can start using MUSIC functions for advanced querying.

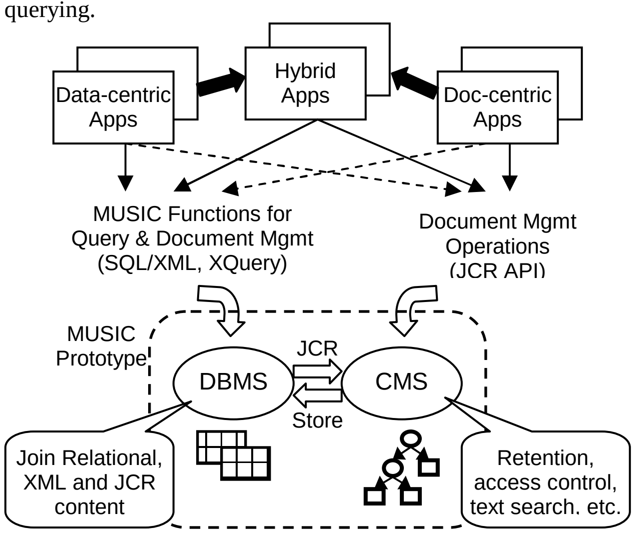
Data-centric Apps
Hybrid Apps
Doc-centric Apps
MUSIC Functions for Query & Document Mgmt (SQL/XML, XQuery)
Document Mgmt Operations (JCR API)
MUSIC Prototype
DBMS
JCR
Store
CMS
Join Relational, XML and JCR content
Retention, access control, text search. etc.

Figure 5: Components of the prototype

4.3 Querying the MUSIC Repository

Advanced query support is provided through the SQL view. As discussed in Section 4.1, the JCR data model supports hierarchical representation of content and its metadata. So, the main challenge is to find an easy and flexible way of modeling JCR hierarchies in the SQL statements. We implemented a table function called JCRTable which achieves this goal. Figure 6 shows the JCRTable syntax. Our solution was inspired by XMLTable function in SQL/XML standard [32].

The JCR-node-query part takes a JCR XPath query and identifies a sequence of nodes in the JCR repository. In the optional **PASSING** section, the user can provide multiple input parameters to be used in the JCR queries. These parameters identify JCR workspaces and connection arguments. Finally, for the each node found, the desired property values are extracted and bound to column names defined in the **COLUMNS** section. The JCR-property-query values for each column can be either a property name, or another JCR XPath query identifying a property

[3] Most importantly, we need unique identification of JCR nodes to be able associate and process them during the query evaluation. For this reason, we enforce every node in the JCR repository to have universal unique identification (UUID) values.

[4] Actually the MUSIC repository can support all kinds of unstructured data. However, we developed special techniques for XML data to take the advantage of the native XML store, indexes, and XQuery language support available in DB2.

relative to the current the node. The located value is cast to the specified type and then bound to a column name so that it can be freely used in the body of the SQL statement. The JCRTable function always outputs a default column called uuid which holds the UUID values of the JCR nodes identified by the input JCR-node-query.

```
JCRTable( JCR-node-query,
   [ PASSING argument-list  ]
   [ COLUMNS
      column-name1 data-type
                    PATH JCR-property-query1,
      column-name2 data_type
                    PATH JCR-property-query2, ... ]
)
```

Figure 6: JCRTable function syntax

Once again consider the real-estate application that uses the JCR tree shown in Figure 4 to store the documents about available houses on real-estate market. This application can offer an elaborate reporting service by using the SQL view of the MUSIC repository to issue queries. For example, the query in Figure 7 obtains the number of real estate listings mentioning roof problems in their inspection reports.

```
SELECT count(*)
FROM JCRTable('/listing[reports[
          text-contains(.,"roof problem")]]')
```

Figure 7: Querying the MUSIC Repository (1)

Notice that the above query includes a JCR text search predicate. In our prototype, the JCR system contains an integrated text search engine. Actually this simple SQL statement is quite powerful. If this application was a pure document-centric one without any advanced query support, obtaining the same result solely using the JCR API would require coding a full Java program. This can be a cumbersome and challenging task, especially when a large number of ad-hoc operations (e.g. roll-up or drill-down) need to be performed over the JCR repository for business analytics.

Continuing the example, further assume that our hybrid real-estate application keeps a history of completed real-estate transactions in a Sales table with MlsNumber, SalePrice and ListDate and SaleDate columns in the DB2 database. Then, the query shown in Figure 8 can be issued against the MUSIC repository to obtain an average number of days that the previously sold houses stayed on the market for the zipcodes of currently available houses. This information can be used for real-estate market analysis like determining whether a specific house is in a high demand neighborhood, and making an informed offer accordingly. Notice that the JCRTable function only locates the XML description property (xmldoc in SQL statement), and main manipulation is performed by XMLTable SQL/XML function.

In terms of implementation, JCRTable function is actually just a syntactic sugar on top of low level MUSIC functions. We have implemented separate functions for obtaining node sequence (JCR-node-query part in JCRTable syntax), and extracting column values for each data type. A list of these functions and a version of the above query rewritten with them are provided in Appendix A.

```
SELECT x.zipcode,
   AVG(s.SaleDate - s.ListDate) AS marketdays
FROM JCRTable('/listing'
              COLUMNS
                 mlsnumber CHAR(8)
                  PATH './mlsnumber',
                 xmldoc    XML
                  PATH './description') AS j,
     XMLTable('$i/detailed_listing'
              PASSING j.xmldoc AS "i"
              COLUMNS
                zipcode    CHAR(5)
              PATH './address/zipcode') AS x,
     sales s
WHERE s.mlsnumber = j.mlsnumber
GROUP BY x.zipcode
ORDER BY marketdays
```

Figure 8: Querying the MUSIC Repository (2)

Querying the JCR Repository with XQuery:

We built another MUSIC function for XQuery, called JCRQuery. The JCRQuery function takes a JCR XPath expression as an input to locate a property in the tree having XML document content. Returned XML document is processed in the XQuery. The query in Figure 9 returns the addresses of the houses with hardwood floors.

```
for $i in fn:JCRQuery('/listing[
 text-contains(.,"Hardwood")]/@description')
return <result> { $i//address } </result>
```

Figure 9: Querying the MUSIC Repository (3)

Similar to JCRTable, the JCRQuery function is implemented with low-level MUSIC functions.

4.4 Document Management Operations in MUSIC Repository

In this section, we present document management support in the SQL view of MUSIC repository. Although the JCR view provides this support to the fullest extent, some hybrid application developers may prefer to use the SQL view as they are already familiar with it. This support may be particularly attractive choice for data-centric applications since they already store and query XML documents, but suffer from the lack of high-level document management functions like check-in/check-out, versioning, foldering operations, etc. We anticipate that this type of applications demand document management operations over their document collections without making major changes. Although the operations mentioned in this section mainly focus on XML documents, they are applicable to other binary and text document types as well.

In the SQL view, input XML documents are stored in columns of database tables. The prototype provides a set of tools and functions for the DBAs and application developers so they can choose which XML columns to be managed, and what document management features they want to apply. After the set up is complete, the desired document management operations can be used transparently against the existing application queries and update statements.

Internally, XML documents are stored in the JCR repository to be able to perform the document management tasks. But, the SQL view provides a table-based conceptual organization so that from the application perspective, documents are inserted, selected and deleted from tables. The current implementation of the MUSIC repository offers functions for basic document management support such as versioning, locking and foldering operations (see Appendix B for the full list). We create a special workspace in the JCR repository whose organization is controlled and managed by the MUSIC system. We store each XML document in a separate node that includes a set of properties for storing the document content, an optional metadata document (in XML, its use is explained later), database, table and column names, and primary key value(s). The XML document and metadata properties are set to be versionable. Additional nodes may be created in the JCR repository to represent folder structure, and perform folder operations.

Configuration:

Our main goal is to configure the database and JCR repository in a way that hybrid application developers can create a SQL view for their managed content. We also have to consider existing data-centric applications that want to add document management features. For this purpose, we developed a configuration tool that the users can either create a new table with a managed XML column, or they can point to an existing table and choose the XML columns for migration. The tool works in three steps:

(1) It creates a new internal table with the specified column definitions (or exactly the original column definitions if it is a migration from an existing table). However, there is an exception for managed XML columns. The type of the selected XML column is changed to character type so that it can now store JCR node references (UUID values). If this is a migration from an existing table, the tool moves the XML documents into the JCR repository, and the rest of the data is loaded into a new internal table along with the associated UUIDs.

(2) The tool creates an SQL view over the internal table. The view has the specified table name and column definitions. The body of the view includes a select SQL statement that returns all the column values from the internal table and corresponding XML documents from the JCR repository with the help of the UUID value and the MUSIC function for document retrieval[5]. Figure 10 illustrates this setup in the MUSIC repository. All the required access control mechanisms provided by JCR can be applied for the XML documents. Optionally, the view may include extra columns to return UUID and folder information for the documents to be able to perform other document management tasks.

(3) To support insert/update/delete operations, the tool creates corresponding instead-of triggers for the view. The trigger bodies include all the appropriate MUSIC functions for inserting, versioning and deleting the documents. For example, the update operation uses check-out and check-in functions. In addition, the users may also want to specify extra columns that need to be managed along with the XML document[6]. In this case, the trigger bodies include suitable SQL/XML constructs for creating an XML metadata document, attaching it to the actual document, and retrieving it with the document as well.

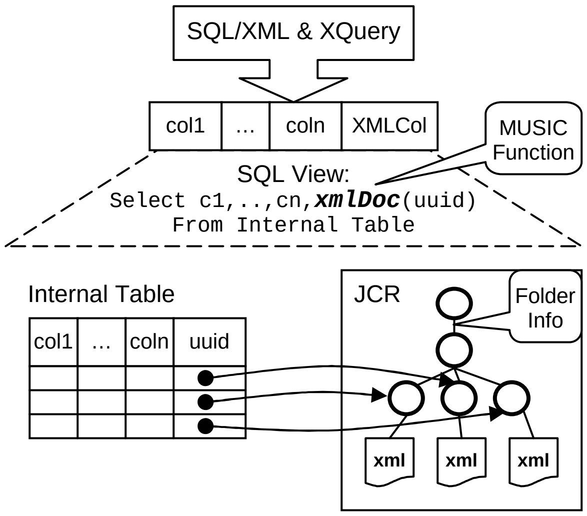

Figure 10: Database and JCR setup for document management operations

The tool outputs a set of scripts which can be directly executed against the database. It is also possible that, the power users could customize the scripts. Instead of utilizing this tool, they can always perform customized configuration and setup operations with the MUSIC functions.

Figure 11 shows a table view for real estate listings over an internal base table as described in Step 2. Then, we can perform document management operations such as inserting a new document with folder information, check-out the document, changing the folder, etc. as shown.

4.5 Improving the System Performance

The DBMS used in the MUSIC prototype (and other major products available on the market) provides native XML storage support for fast retrieval and indexing. Unfortunately, the queries issued to the MUSIC prototype cannot leverage this support since all the documents are owned by the JCR system. We overcome this problem by developing effective caching mechanisms for XML documents inside the DBMS.

Figure 12 shows the main organization of the XML cache. We created an internal table with a native XML column. This table also includes columns for UUID, version, timestamp and property name values to be able locate documents in the JCR repository. MUSIC system administrators can create tailored XML indexing and clustering strategies for this table to tune it according to the application characteristics.

Populating the Cache:

Cache table population occurs in different ways depending on which view interface is used. For every document operation

[5] This function internally uses the same retrieval function implemented for JCRTable (See the Appendix for more details).

[6] There might be cases where the values of these columns could be related to the XML document. For example, they could be extracted from the document for fast retrieval. So, versioning only the XML document may run into consistency problems.

```
CREATE VIEW Listing(mlsnum, price,
          doc, docid, folder) AS
  SELECT l.mlsnumber, l.price,
          music.xmlDoc(l.docid), l.docid,
          music.getfolder(l.docid)
  FROM Listing$music l;
```

```
INSERT INTO listing(mlsnum, price, doc, folder)
VALUES('1', 200000,
xmlparse(document '<a><b>some text</b></a>'),
                         '/mlssales/onsale');
```

```
SELECT music.co(docid)
FROM listing
WHERE mlsnumber='1';
```

```
SELECT doc,folder
FROM listing
WHERE mlsnum = '1';
```

```
SELECT music.movedoc(l.docid, '/mlssales/sold')
FROM listing l
WHERE mlsnumber='1';
```

```
xquery
  for $i in db2-fn:xmlcolumn('LISTING.DOC')//b
  return <res>{$i}</res>;
```

Figure 11: Example Document Operations and Queries

(insert, modify or delete), the JCR repository asynchronously updates its text index with the modified content. We modified this mechanism to move input XML documents from JCR repository to the native XML store in the DBMS. When the JCR view is used to insert/update/delete XML documents, we capture the updates this way and apply them onto the cache table in the DBMS. The cache table will eventually contain all the XML documents in the JCR repository. Alternatively, it may also be populated on demand, as the users' requests are fulfilled.

When the SQL view is used, the cache table is populated and managed as the applications issue their SQL commands. Inside the MUSIC functions, required JCR operations are performed upfront to obtain the column values for the cache table. The documents manipulated this way are marked at the JCR repository to skip cache maintenance mechanism for JCR views.

Cache Utilization:

The core MUSIC function for XML document retrieval (`jcrXML` in Appendix A[7]) is implemented to take the advantage of the cache. This function first checks the cache table for a given node UUID, version timestamp and property name values. There might be cases where new XML documents may not appear in the cache yet (the JCR repository text maintenance mechanism is asynchronous). For such cases, the JCR repository is accessed to return the document.

5. RELATED WORK

Integration of diverse information systems has a long history. In particular, integration of structured and unstructured data models and query languages has been independently studied in the DB and IR literature. The database community proposed many text

[7] Remember that this function is internally used by the `JCRTable` function and it can be directly used by hybrid applications.

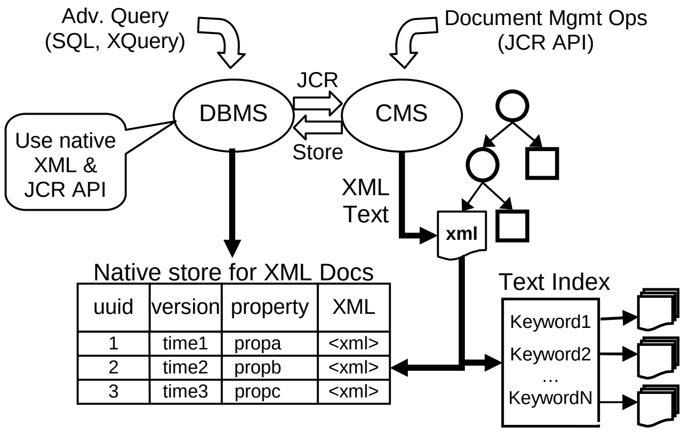

Figure 12: XML is text indexed, and stored natively in DBMS

extension schemes that merge ranked result retrieval with regular execution methods. All the major RDBMS products today provide full text query support [17]. The IR literature, on the other hand, includes numerous proposals implementing IR methods over DBMSs (e.g. [26][27]). A common theme in this research is to exploit underlying database mechanisms to enable better unstructured search. The recently introduced XML standard has further fueled DB+IR integration efforts in both communities by shrinking the gap between structured and unstructured data [5][6][16][18][25]. Thorough discussions on DB+IR integration work and a good coverage of literature can be found in [4][5][9][16][17].

As we described before, all these efforts analyzed and attacked the integration problem mainly from the querying perspective. In this paper, we are suggesting a broader view to integration approach by considering all aspects of DBMS, CMS and IR together. The ever increasing market share for enterprise CMS products is clear evidence that enterprise applications are demanding a new set of information management features in addition to vital querying support. As these features interact with querying mechanisms, structured and unstructured data integration needs to be investigated by taking the entire feature set into account.

[32] also compares and contrasts data-centric and document-centric paradigms. It also identifies the technical challenges and opportunities for bringing these different paradigms closer together. However, contrary to our proposal, the authors claim that there cannot be a single system that can manage both structured data and content equally well and in a scalable manner. Hence, they suggest an integration layer on top of loosely coupled DBMS and CMS repositories. Although this is similar to our prototype model, our grand vision is to build a unified repository model that owns and manages the integrated content.

Our prototype approach for the mapping JCR data model into relational is similar to other hierarchical modeling efforts in information integration systems. In fact, our solution was inspired by the XMLTable function in the SQL/XML standard [33]. Similar approach was also proposed in [19] where XML collections are queried by SQL queries.

The recently introduced Oracle 10g product stepped towards our proposed integration model by blending XML management features with its content management functionalities [31]. It allows inserting an XML document into a folder through a WebDAV interface, and issuing XQuery statements to document collections with a database view. Although available document management features are very limited, we anticipate that this move will change the expectations among hybrid enterprise application developers.

6. CONCLUDING REMARKS

In this paper, we argued that a new class of enterprise applications are emerging due in large part to increasingly complex business requirements such as Sarbanes-Oxley Act and the widespread adoption of XML for the representation of all types of complex business data. They have the analytic requirements of data-centric applications and the document lifecycle and metadata management requirements of document-centric applications to process structured and unstructured integrated content. Today, neither DBMS nor CMS technologies can answer the growing demand of management operations for the integrated content.

We explored the trends giving rise to these hybrid applications and examined their requirements. We argued for a holistic approach to design a unified repository solution. We discussed research challenges that must be met in order to achieve an enterprise scale solution. Finally, we detailed our early experience in this space by presenting a repository prototype developed as part of a project we call MUSIC (Management of Unstructured and Structured Integrated Content).

ACKNOWLEDGMENTS

We would like to thank Andrey Balmin, Volker Markl, Kevin Roundy and Malte Randt for their help and insights on the MUSIC project.

REFERENCES

[1] 2nd Int. Conf. on the Clinical Document Architecture, Oct., 2004, Acapulco, Mexico, http://www.hl7.de/iamcda2004/.

[2] Adobe Intelligent Document Platform, http://www.adobe.com/enterprise/idp.html.

[3] R. Agrawal, C. Johnson, J. Kiernan, F. Leymann, “Taming Compliance with Sarbanes-Oxley Internal Controls Using Database Technology”, ICDE 2006.

[4] S. Amer-Yahia, Moderator, “Databases and Information Retrieval: Rethinking the Great Divide”, Panel, SIGMOD 2005.

[5] S. Amer-Yahia, L. V. S. Lakshmanan, S. Pandit, “FleXPath: Flexible Structure and Full-Text Querying for XML”, SIGMOD Conference 2004: 83-94.

[6] S. Amer-Yahia, N. Koudas, A. Marian, D. Srivastava, D. Toman, “Structure and Content Scoring for XML”, VLDB 2005: 361-372.

[7] R.A. Baeza-Yates, M. P. Conses, “The Continued Saga of DB-IR Integration”, Tutorial, VLDB 2004.

[8] K. S. Beyer, et al., “System RX: One Part Relational, One Part XML”, SIGMOD Conference 2005: 347-358.

[9] M. J. Carey, “BEA Liquid Data for WebLogic: XML-Based Enterprise Information Integration”, ICDE 2004

[10] S. Chaudhuri, R. Ramakrishnan, G. Weikum, “Integrating DB and IR Technologies: What is the Sound of One Hand Clapping?”, CIDR 2005: 1-12.

[11] D. M. Choy, “Integration of Structured and Unstructured Data in IBM Content Manager”, SIGMOD 2005

[12] Day Software, www.day.com/.

[13] DB2 Content Manager, www.ibm.com/software/info/contentmanagement/.

[14] DB2 UDB v9.1 Beta, www.ibm.com/software/data/db2/xml/.

[15] EMC Documentum, www.documentum.com/.

[16] N. Fuhr, K. Grossjohann, “XIRQL: An XML query language based on information retrieval concepts”, ACM TOIS 22(2), 2004.

[17] L. Gravano, Editor, Special Issue on Text and Databases, Data Eng. Bulletin, December 2001 Vol. 24 No 4.

[18] L. Guo, F. Shao, C. Botev, J. Shanmugasundaram, “XRANK: Ranked Keyword Search over XML Documents”, SIGMOD 2003.

[19] A. Halverson, V. Josifovski, G. Lohman, H. Pirahesh, M. Mörschel, “ROX: Relational Over XML”, VLDB 2004: 264-275

[20] T. Hartman, “The Cost of Being Public in the Era of Sarbanes-Oxley”, Foley & Lardner LLP NDI Study, June 2005.

[21] HL7 Clinical Document Architecture (CDA) Specification, www.hl7.org/v3ballot/html/infrastructure/cda/cda.htm.

[22] IBM WebSphere Business Integration Server Foundation, www.ibm.com/software/integration/wbisf/

[23] IBM Workplace Forms, http://www.ibm.com/software/ workplace/products/product5.nsf/wdocs/formshome.

[24] Java Content Repository (JCR) Standard, www.jcp.org/en/jsr/detail?id=170.

[25] R. Kaushik, R. Krishnamurthy, J.F. Naughton, R. Ramakrishnan, “On the Integration of Structure Indexes and Inverted Lists”, SIGMOD 2004.

[26] I. A. Macleod, R. G. Crawford, “Document Retrieval as a Database Application”, Information Technology: Research and Development, 2, 43-60, 1983.

[27] S. Melnik, S. Raghavan, B. Yang, H. Garcia-Molina, “Building a distributed full-text index for the web”, 10th Intl. World Wide Web Conf. (WWW10), pp. 396–406, May 2001

[28] Microsoft Enterprise Content Management White Paper, www.microsoft.com/office/preview/ecmwhitepaper.mspx.

[29] Microsoft Office 2003 XML Reference Schemas, www.microsoft.com/office/xml/default.mspx.

[30] The Java Transaction API (JTA), java.sun.com/products/jta/index.html.

[31] Oracle Database, www.oracle.com/database/.

[32] A. Somani, D. Choy, J. C. Kleewein, “Bringing together content and data management systems: Challenges and opportunities”, IBM System Journal, Vol. 41, No. 4, 2002

[33] SQL/XML Standard, www.sqlx.org/.

[34] Unstructured Information Management Architecture (UIMA), IBM Research, www.research.ibm.com/UIMA/.

[35] WebFountain Project, IBM Research, www.almaden.ibm.com/webfountain/.

[36] WebSphere Portal for Multiplatforms, www.ibm.com/software/genservers/portal/.

[37] The World Wide Web Consortium, “XForms – The Next Generation of Web Forms”, www.w3.org/MarkUp/Forms/.

[38] The World Wide Web Consortium, “Semantic Web”, www.w3.org/2001/sw/.

[39] XBRL—Extensible Business Reporting Language, www.xbrl.org/.

[40] ZDNet News, news.zdnet.com/2100-9592_22-5227856.html.

APPENDIX

All the SQL examples provided in this section are developed using IBM DB2 DBMS syntax.

A. MUSIC Functions for querying JCR Repository

- Get the root of a workspace:
  ```
  jcrRoot (connect args) returns node
  ```
- Get a table of node references (UUIDs) from a context and a JCR query:
  ```
  jcrNodes (uuid, 'JCR-node-query')
                       RETURNS TABLE(uuid)
  ```
  Get a table of node references using the default workspace settings:
  ```
  jcrNodes ('JCR-node-query')
                       RETURNS TABLE(uuid)
  ```
- Get a scalar SQL value from property of a node identified by its UUID:
  ```
  jcrInt (uuid, 'JCR-property-query')
                             RETURNS INTEGER
  jcrDouble (uuid, 'JCR-property-query')
                             RETURNS DOUBLE
  jcrString (uuid, 'JCR-property-query')
                             RETURNS VARCHAR
  jcrDate (uuid, 'JCR-property-query')
                             RETURNS DATE
  jcrXML (uuid, 'JCR-property-query')
                             RETURNS XML
  jcrBLOB (uuid, 'JCR-property-query')
                             RETURNS BLOB
  jcrCLOB (uuid, 'JCR-property-query')
                             RETURNS CLOB
  ```

The following is a verbose form of the query provided in Section 4.3, rewritten with above MUSIC functions:

```
SELECT x.zipcode,
   AVG(s.SaleDate - s.ListDate) AS marketdays
FROM TABLE(music.jcrNodes('/listing')) AS node,
     XMLTable('$i/detailed_listing'
         PASSING
          music.jcrXML(node.uuid, 'description')
              AS "i"
         COLUMNS
           zipcode    CHAR(5)
              PATH './address/zipcode') AS x,
     sales s
WHERE s.mlsnumber =
         music.jcrString(node.uuid, 'mlsnumber')
GROUP BY x.zipcode
ORDER BY marketdays
```

B. MUSIC Functions for Document Management

Main CRUD operations:

- Document retrieval functions for a given UUID. As the organization of JCR repository is controlled by the MUSIC system, property names are known by the system. Hence, these functions internally exploit corresponding above JCR MUSIC functions to return XML, CLOB and BLOB values.
  ```
  xmlDoc (uuid) returns XML
  textDoc (uuid) returns CLOB
  binDoc (uuid) returns BLOB
  metadata(uuid) returns XML
  ```
- Inserting a new document into MUSIC repository
  ```
  insertXML (doc XML [, 'folder string',
             metadata-doc])        RETURNS uuid
  insertText (doc CLOB [, 'folder string',
             metadata-doc])        RETURNS uuid
  insertBin (doc BLOB [, 'folder string',
             metadata-doc])        RETURNS uuid
  ```
- Versioning operations:

  Check-in, check-out, and uncheck-out:
  ```
  ci(uuid, document, metadata) RETURNS TIMESTAMP
  co(uuid) RETURNS INTEGER
  unco(uuid) RETURNS INTEGER
  ```
  Restore a version identified with a timestamp
  ```
  restore(uuid, version-timestamp) RETURNS uuid
  ```
  Get the current version of the document
  ```
  baseversion(uuid) RETURNS TIMESTAMP
  ```
  Get all the versions of a document
  ```
  versions(uuid) RETURNS TABLE(TIMESTAMP)
  ```
- Putting and removing long duration locks on the document.
  ```
  lock(uuid) RETURNS INTEGER
  unlock(uuid) RETURNS INTEGER
  ```
- Delete the document from JCR tree.
  ```
  delete(uuid) RETURNS INTEGER
  ```

Folder operations:

- Create a new folder in the JCR tree.
  ```
  newfolder('folder-path') RETURNS INTEGER
  ```
- Delete a folder.
  ```
  delfolder('folder-path') RETURNS INTEGER
  ```
- Assign or change the folder of a document.
  ```
  movedoc(uuid, 'folder-path') RETURNS INTEGER
  ```
- Get folder information of the document.
  ```
  getfolder(uuid) RETURNS VARCHAR
  ```
- Find all the documents in a folder.
  ```
  folderdocs('folder-path') RETURNS TABLE(uuid)
  ```
- Lock/unlock a folder. All the sub-folders and documents are affected with this operation.
  ```
  lockfolder('folder-path') RETURNS INTEGER
  unlockfolder('folder-path') RETURNS INTEGER
  ```
- Move a folder to another folder.
  ```
  movefolder('folder-path', 'new-path' )
                                 RETURNS INTEGER
  ```