Bringing together content and data management systems: Challenges and opportunities

by A. Somani
D. Choy
J. C. Kleewein

With advances in computing and communication technologies in recent years, two significant trends have emerged in terms of information management: heterogeneity and distribution. Heterogeneity (herein discussed in terms of different types of data, not in terms of schematic heterogeneity) pertains to information use evolving from operational business data (e.g., accounting, payroll, and inventory) to digital assets, communications, and content (e.g., documents, intellectual property, rich media, e-mail, and Web data). Information has also become widely distributed, both in scale and ownership. To manage heterogeneity, two major classes of systems have evolved: database management systems to manage structured data, and content management systems to manage document and rich media information. In this paper, we compare and contrast these different paradigms. We believe it is imperative for any business to exploit value from all information—independent of where it resides or its form. We also identify the technical challenges and opportunities for bringing these different paradigms closer together.

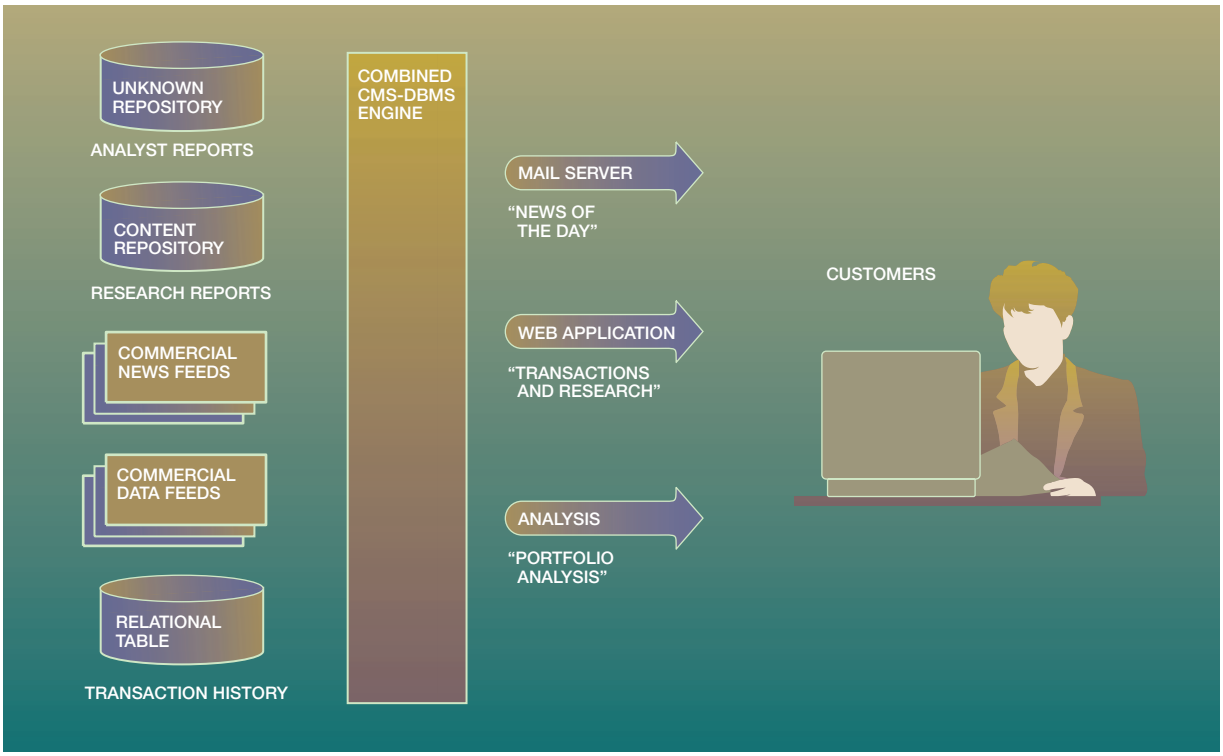
Database management systems (DBMSs)—in particular, relational database management systems (RDBMSs), such as the IBM DB2 Universal Database* [1] (Database 2*)—have long been established as the appropriate engines for providing transactional and analytical processing over homogeneous, well-typed, structured information, while providing extensibility through object-relational extensions. Recently, content management systems (CMSs), such as the IBM Content Manager,[2] have emerged to manage non-traditional, heterogeneous, unstructured data (e.g., documents, images, and rich media).[3–5] CMSs provide for higher-level semantics, such as versioning, foldering, check-in/check-out, and meta-data management. Although a CMS typically uses an RDBMS internally to manage the meta-data that describe the unstructured data, a CMS is usually designed as a self-contained system to manage content.

Increasingly, it is becoming important that information—regardless of its format, source, and location—needs to be easily managed, searched, and accessed. While these seemingly disparate systems continue to evolve, they must become more integrated so that business applications can be easily implemented across all information. To this end, a system that can provide integrated access to a federation of distributed, heterogeneous information systems is needed. Let us explore the need for such a system with the help of the following scenario.

Consider an on-line brokerage that provides its customers with on-line trading capabilities, such as the ability to submit new transactions, view transaction histories, manage unexecuted transactions, check quotes, and view account balances. In addition, most brokerages provide other value-added services, such

 0018-8670/02/$5.00 © 2002 IBM

Figure 1 Brokerage scenario



as access to research reports, complex charting and comparison functions, and access to current news.

A simplified view of such a data topology is depicted in Figure 1.

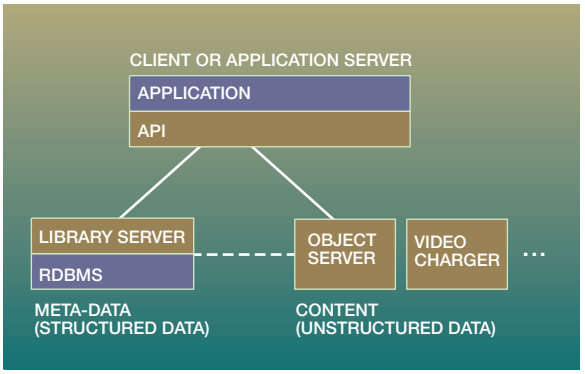
Data come from a number of different sources, including traditional transaction data (such as stock market trades and account balances), rich media repositories (such as those that hold company reports), real-time data feeds (such as stock ticker data and news feeds, which may themselves be rich media), and repositories whose exact format is not known because they are not owned by the brokerage and only export data through services (such as Web services).

From such a topology, information is delivered to the customer in the form of mail (such as "news of the day about companies in my portfolio"), Web applications (such as on-line trading applications), and complex analytic applications (such as portfolio analysis and charting). In such a topology, data from multiple sources are required to satisfy any of these functions, and those data might be digital media or simply transactional. For example, "news of the day" requires access to transactional portfolio data to determine what news from a live feed is interesting, and portfolio analysis requires access to transactional portfolio data, as well as company and analyst reports about the securities in the portfolio.

In addition, complex analytics of the data, rich media or otherwise, may be required to provide additional services. For example, a stock search may be of the form "show me stocks with a five-year CGR (Compound Growth Rate) of 10 percent or more, with recent positive news reports, where 50 percent of the analysts covering are bullish on the stock." Neither a traditional transactional DBMS nor a traditional CMS can process such a search.

In this paper, we compare and contrast these two classes of systems with the objective of integrating the two. In the next section, we start out by discussing the background and historical evolution of these

Figure 2 Content Manager



systems and offer a high-level framework for comparison. We then explore specific areas of differentiation in greater depth. The section after that proposes an "information integration" architecture. Finally, we conclude with some technical challenges.

Background

When investigating the convergence of rich media systems with transactional systems, it is important to first understand the differences in the purpose of those systems in the following key areas:

- Initial target problem domain
- Typical use
- Type of data and their associated semantics
- Functionality required

An RDBMS is designed to manage business records, providing an on-line transaction processing (OLTP) capability to support business operations and a data analysis (on-line analytical processing, or OLAP) capability to support decision-making. Transactional systems have their background in general ledger applications and business data management—managing and manipulating data electronically that would otherwise have been stored in a ledger or spreadsheet. Industries that have in the past relied on ledgers are major users of these kinds of systems. One example of such an industry is banking, which relies on accounts, transaction histories, and a long list of debit and credit operations.

Ledgers and spreadsheets typically hold very granular data such as account codes, names, account balances, and dates of shipments. The goal of such systems is to manipulate and analyze the exact data of interest and to present the results of those operations to a user. These data values tend to be heavily numeric-, date-, and short-character-string-based and do not typically have a lot of natural-language text or rich-media data. The users of such a system tend to be applications or machines (automatic teller machines [ATMs], point of sale [POS] terminals, etc.), often from more accounting or data-facing domains and with strong analytic requirements. OLTP operations are typically synchronous, short-duration transactions. Once transactions are complete, the DBMS provides decision-support querying and other analytic processing to make business decisions. The performance of a system is often measured by its throughput: the number of transactions or queries it can process per second.

In contrast, a CMS, such as the IBM Content Manager (Figure 2), has its background in business document management—managing and tracking documents electronically that would otherwise be stored in a library or filing cabinet. Libraries and filing cabinets typically hold entire documents or parts of documents, and the goals of such systems are to identify documents that may contain information of interest and to present a list of such documents to users. These systems have evolved over time to support digital assets in many different forms. Users in industries that have in the past relied heavily on pictures or paper forms with large areas of free-form text are major users of such systems. One example is the insurance industry, which relies on documents, such as policies, claims forms, photos of damage, and accident reports.

CMSs have been used both as on-line repositories for operational data and as archival libraries for long-term retention. Thus, a CMS must support the complete life cycle of content entities, from creating to updating, organizing, searching, accessing, distributing, tracking, and retention. To do so, a CMS also needs to manage a variety of meta-data that describe these content entities. Meta-data include system attributes, such as a time stamp and content entity size, that the CMS uses to manage content. Meta-data also include application attributes, such as author, subject, and keywords, that an application uses to describe content entities. Frequently a CMS must also support business processes (e.g., workflow). Since content is pervasive, the user of CMS data can be anyone and anywhere. As a result, content applications are usually network-centric and often interactive in nature, perhaps along with a high-volume batch in-

gestion operation. In terms of data representation, a CMS supports higher-level semantics for content, which needs to be instance-based for access, manipulation, and control. Some applications need a complex structure to represent each instance. Hence, a CMS needs more than the simple tabulated format (i.e., a collection of homogeneous, flat records) used by an RDBMS for data modeling. It must maintain various relationships among instances. Aggregations (e.g., folders) are often needed to represent collections of heterogeneous instances, such as documents of different kinds as well as other folders.

Because of their origins in business document management, these systems tend to deal with objects that are "user viewable" and that often have large components written in a natural language. Search capability is very important to users of these systems, including both attribute searching and text searching. Because large amounts of unstructured data are involved, an update transaction can be fairly long compared to that of a transactional system. Furthermore, a CMS must handle asynchronous operations to accommodate content streaming and delivery to or from a third party, to enhance performance (for large objects and parallelism), and to improve availability. The performance of a system is often measured by its response time for returning the first set of results of information, either the beginning of a large result set or the beginning of a large document. For a high-volume production system, a good throughput rate for loading unstructured data (e.g., images) is also essential.

We believe that information sources will continue to be heterogeneous and distributed.

Meanwhile, object-relational (OR) technology has emerged from the RDBMS community in an effort to extend the relational model to accommodate unstructured data. However, OR-DBMSs have not been deployed as a full CMS in the marketplace to support large-scale content applications, and the following are a few of the reasons:

- OR-DBMS implementations are mostly based on existing general-purpose RDBMSs, which do not have sufficient system capabilities and the infrastructure needed to handle a large volume of unstructured data. Examples are hierarchical storage management function, policy-based storage administration, content retention management, separate delivery path for unstructured data (to meet network topology, bandwidth, streaming, and quality-of-service requirements), and support for asynchronous operation.
- The object-relational data model is not yet rich enough to capture the high-level semantics of content. Although the relational model is built on a homogeneous set of records, a CMS data model is typically built on a document instance and heterogeneous collection, plus a meta-data model. (See later subsection on the data model.) Without the concept of higher-level entities (documents, folders), an OR-DBMS is unable to maintain the integrity of such entities, to provide suitable access control for them, and to offer high-level content functions that enhance both usability and performance. For example, a CMS may provide a function for retrieving a folder (and its meta-data) together with all the documents (and their respective meta-data) that are contained in the folder. To accomplish the same task through an RDBMS or OR-DBMS, a user has to understand the database design and possibly issue many Structured Query Language (SQL) queries.

The trade-off between generalization and specialization of these systems will be a challenge. We believe that a DBMS will continue to support and enhance OLTP and OLAP functions, whereas CMS will focus on managing a large volume of content with the high-level semantics needed by its applications. In the foreseeable future, we do not believe that there will be a single system that can manage both structured data and content equally well and in a scalable manner.[6] Instead, we believe information sources within an enterprise will continue to be heterogeneous and distributed.[7] The challenge, then, is how to integrate such information.

The industry has come up with various federation alternatives to handle separately managed data. One style of federation has been across homogeneous, structured data[8,9] with full-scale, composite query optimization, planning, and execution, along with strongly defined schemas and transactional semantics. Over the years, several systems have been developed to deal with heterogeneous database systems (Garlic,[10] DISCO,[11] and TSIMMIS[12,13]). Content sys-

Table 1 Summary of differences between a CMS and a DBMS

	Content Management	Data Management
Initial problem domain	Library and filing cabinet	General ledger and spreadsheet
Who uses it	Ordinary people, work group	Applications, specialists
Type of data	Human readable with heavy text or image	Strongly data focused, heavily computational
Data granularity	Entire document or document fragment	Individual data values
Usage scenario	Archival and retrieval	Analytic, transactional application
Search task	Parametric and text, to find documents	Mostly parametric with text extension, to find data values
Typical data set sizes	~O(petabytes)	~O(terabytes)
Performance metric	Time to first response (user interactions)	Transactions/queries per second (TPC-C**, TPC-H**)

tems, in contrast, have provided unified access to heterogeneous information[14] with broadcast ("union") searching over loosely defined schemas across distributed sources. We discuss differences in federation in more detail later.

Thus, the two kinds of systems are designed for very different problem domains, supporting different applications and different users. They manage data of different types, granularity, representation, and semantics and behavior. They offer different functions, and the access pattern and transaction model are quite different. The differences in the two systems can be summarized in Table 1.

Detailed comparisons of differences between a CMS and a DBMS

Although these systems initially addressed different problem domains, that is no longer the case. Customers increasingly demand consistent, high-function access to all information assets, wherever they reside and however they are stored. They no longer want merely a database management system; they require an integrated information system.[15]

There are many properties such a system must exhibit at the technical level to satisfy the needs of the CMS market as well as the DBMS market. At the highest level, it must offer consistent functionality across a wide variety of existing data sources. Clearly the scope of this effort is tremendous, and a detailed investigation into all aspects of such an effort is well beyond the scope of this paper. Rather than merely mention the required areas of convergence, however, we focus on the problem of integration across heterogeneous data. Hence, our detailed comparison focuses on three key data-access technologies:

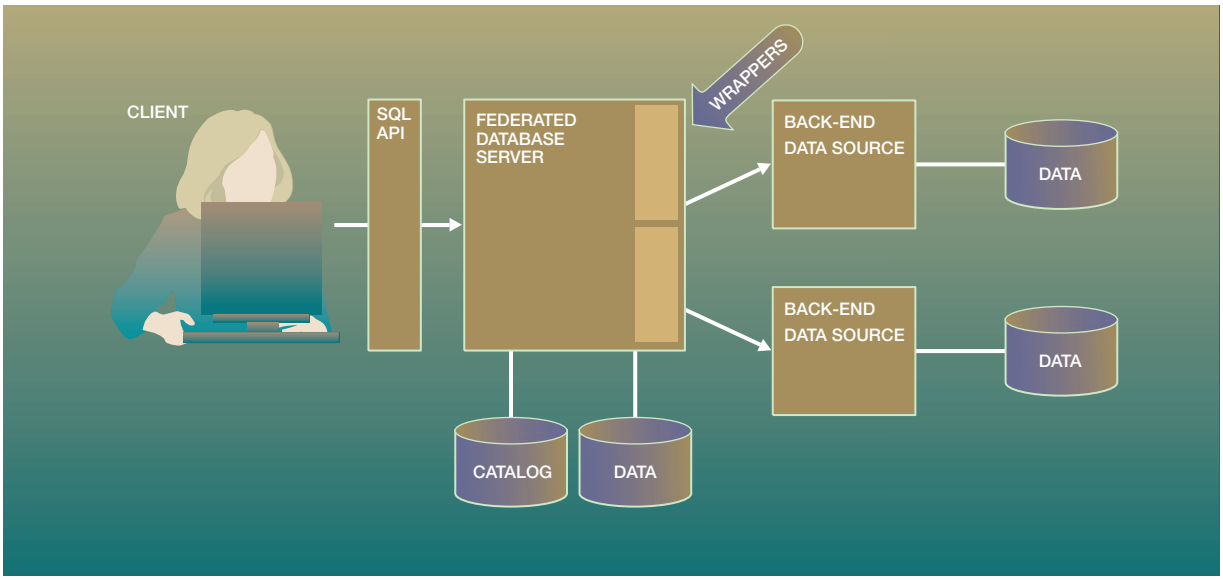
- Data federation to provide in-place access to existing data
- An expressive data model that accommodates data from very disparate sources
- Search over meta-data and data

Federation. Data federation is one technology that can deliver consistent, high-function access to existing data.[16] Transactional systems[9,17] and rich media systems[14] currently offer different data federation capabilities that are well-suited to the historical use of those respective systems. These differences can be characterized in terms of the data that are being searched, the functions and capabilities exposed through that search, and the amount of precision required in the search.

A data federation solution for a transactional system, such as the IBM DataJoiner* multidatabase server, is designed to search transactional data and federate across transaction systems (Figure 3). As a result, this type of solution is focused on queries over very fine granules of data stored in transaction systems. Queries typically are used not only for search but also aggregation, summarization, and complex analytic capabilities, combining data from multiple locations in complex ways and supporting join-type operations in addition to union-type operations. Because of the strong transactional history, this solution produces exact results. True to its spreadsheet and ledger heritage, it must produce results that would survive an audit.

To perform this type of federation and because it is necessary to map data to very fine data granules, a transactional system requires very detailed information about the schema and types of data in the sources to be federated. It must also have a detailed understanding of the semantics of the data sources across a wide

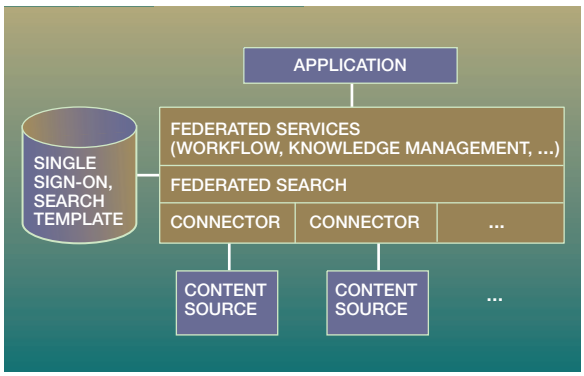
Figure 3 DBMS federation



variety of subject areas (concurrency control, interrogation language, transaction model, etc.) to provide detailed data access. Because the data granules used in these operations are relatively small, care is taken to push operations to data sources to reduce data transfer. (Because the applications are analytic, the user typically sees the result of the operations, not the source data.)

A data federation solution for content systems, such as the IBM Enterprise Information Portal (EIP) (Figure 4), is designed to federate document and rich media systems. The design of EIP is based on an extensible architecture of connectors, offering a loose federation of heterogeneous content sources through a common programming environment. The federation model is to provide a single-search application programming interface (API) for multiple, usually unrelated, content sources. EIP is built on a card-catalog paradigm where searching over meta-data that describe the content is crucial. The search that is supported is more for document discovery. An EIP federated query primarily performs a union operation to combine the search results obtained through native connectors—those backend stores included in the search scope of that query. The "join" operation, as offered by RDBMSs, is not supported by EIP. The advantage of the "union" search is that the engine can start streaming results as soon as they become available. This is especially good for federating over

Figure 4 Enterprise Information Portal



systems that are not always available or have different latencies to various query requests. Overall, searching is more fuzzy in a CMS, and the system is designed to withstand the rigors of human interaction. We explore search in more detail later.

Thus, the differences between content federation and transactional federation can be summarized in Table 2.

Data model. In this subsection, we describe the two data models, one transactional and the other content.

Table 2 Differences between content federation and transactional federation

	Content Federation	Transactional Federation
What is searched	Meta-data and content data	Data
Federated schema	Loosely defined	Strongly typed and defined
What functions are provided	Union search "Loosely coupled" parametric, text and image search	Join, aggregation, analytic Numerical analysis and search
Required precision	Fuzzy	Exact
Transactional semantics	ACID (atomicity, consistency, isolation, durability) within a source (or whatever the data source supports)	ACID across sources with support for two-phase commit (2PC)†

†This presumes that each data source supports ACID by itself and can participate in a two-phase commit.

Transactional data model. The data in a transactional system is modeled around the widely understood relational model.[18] In a relational database, such as one managed by DB2, each relation is a collection of records of the same schema and is represented in the form of a table. Each row in this table is a record, and each column a specific attribute. The schema for a relation is simply a predefined set of attribute names and their type information. In contrast, in a hierarchical database, such as one managed by the IBM Information Management System (IMS*), the record structure defined by a schema is a hierarchy of attributes.

Regardless of the record structure, a transactional system is designed to selectively store, filter, aggregate, search, retrieve, update, and delete business records and homogeneous sets of records efficiently. Such business actions are frequently executed programmatically.

Content data model. Content data models are designed to manage unstructured or semi-structured data, which are typically consumed by a person. Early content management systems, such as the IBM ImagePlus* system, were designed to manage a large volume of scanned image documents. To facilitate document search and to support business activity, application-defined meta-data in the form of a set of attributes are maintained to describe each document.[19] To group documents for filing and processing (e.g., workflow) purposes, a folder paradigm is used to represent *ad hoc* collections of documents and other folders, regardless of their type. Later systems, such as the IBM Content Manager, were designed to manage text documents and digital media assets as well. Such systems require a more flexible data model for meta-data, with user specifiable schema; a content search capability, especially full-text search; and versioning support. Other systems, such as Lotus Notes*, were designed to manage structured documents, with unstructured data captured as document attributes. For some applications, it is not necessary to maintain a distinction between "content" and "meta-data."

Regardless of their intended applications, content management systems typically support a document-centric[20] data model (in terms of handling, manipulation, and access control) that accommodates unstructured or semi-structured data and offers full-text search, versioning support, and collection of heterogeneous documents (as well as collection of collections). The structural aspect of the data model (often for the meta-data) usually varies from system to system.

Search. In the case of a federated DBMS, the primary access and query mechanism is declarative—essentially SQL with object-relational extensions for invoking table functions and other user-defined functions. Query processing is based on the notion of a "mediator" architecture.[21] In the setup phase prior to query processing, each remote data source is modeled as a "local relational view" (called a NICKNAME in DB2); the appropriate wrappers that implement the native query and access mechanism of the respective data source are deployed. In DB2, when the application submits an SQL query to the federated engine, the query compiler in turn cooperates with one or more wrappers written for the remote data sources to come up with an optimized execution plan for the query. If a native data source does not support a particular relational algebra type operation (such as select, project, join, etc.) the federation engine can compensate for it (for example, a file system may not support selection predicates such as "date $<$ 1/1/2002," and the engine could provide for

Table 3 Differences between a CMS search and a DBMS search

	CMS Search	DBMS Search
Search	Meta-data and data	Mostly data
Query mechanism	Primarily procedural	Declarative (SQL with OR extensions)
Global query planning and optimization	No	Yes
Compensation	No	Yes
Results returned	Synchronous plus asynchronous	Synchronous

that). The end applications gain consistent functionality across all data sources; however, the trade-off is that each wrapper needs to be much more sophisticated.

In contrast, in a federated content system such as EIP, the primary query mechanism is procedural. There are APIs to enable search and access across the various back ends. Data entities are represented using the Dynamic Data Object (DDO) of the Object Management Group (OMG), plus the Extended Data Object (XDO) to handle unstructured data. The remote data sources are modeled as DDO and XDO sources. An administrator can create federated search templates, each of which defines a search scope (the set of back-end sources to search) and a mapping of the schema to the native data model of each back-end source. The mapping information is maintained by the EIP federation engine. An application program builds a query expression that can be a combination of a key-value pair style parametric search (e.g., PE $<$ 30 && sector = "Health Care"), text search, and image querying on top of the federated schema. The federated engine in turn translates the query into the native query language for each of the underlying data sources without any attempt at query planning or optimization across the various data sources.

The federated connector uses persistent identifiers (PIDs) as the naming scheme to identify content entities across the heterogeneous back ends. Each PID is a native content ID along with a back-end source ID. The result of a federated query is a list of PIDs that can be returned synchronously or asynchronously. This is in contrast to transactional federation systems in which the results are returned synchronously. The application can then use the PIDs to retrieve the actual objects of interest. This two-step approach turns a heterogeneous result set into a homogeneous one and makes a large result set more manageable. Search in CMSs is also similar: an application first finds the objects of interest through the meta-data repository and then retrieves or manipulates the objects by directly talking to the appropriate content servers.

In a CMS, text search plays a much more important role than traditional DBMS-style analytics on the parametric data (Table 3). Often, the contents of documents cannot be programmatically interpreted and are returned primarily for viewing purposes. Care is taken to limit the number of objects returned to the user by filtering on the meta-data that describe the content.

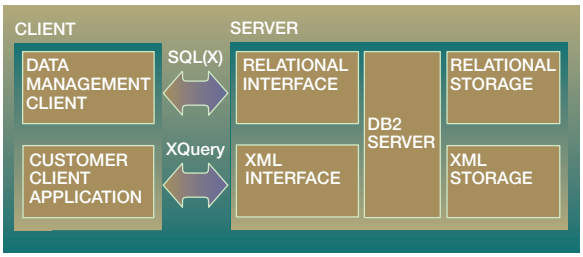
Information integration architecture

Clearly a major challenge of an integrated information system that unifies the CMS and DBMS domains is providing a consistent and powerful architecture for such a system.

With the advent of the Extensible Markup Language (XML)[22] and related standards, structured and semi-structured information can now be represented in an application-independent and nonproprietary format. With the flexibility of XML, together with the expressiveness and modeling capability of XML Schema and the querying power of the XQuery language, XML not only allows information in different representations to be exchanged between applications in a generic format, it also offers an opportunity to establish a framework for integrated access to information managed by heterogeneous systems, including DBMSs and CMSs.

We see an "information integration" system that offers two sets of interfaces to access an open federation of heterogeneous data stores, each built around a standard: a set of XML-based APIs and a set of SQL-based APIs. In spite of the growing popularity of XML, we expect the popularity of SQL to continue because of the simplicity and formal basis of the relational model, its wide acceptance in the marketplace, the maturity of relational technology, and the

Figure 5 Information integration architecture



availability of tools. As discussed earlier, the basis of the federation system is a wrapper architecture in which a wrapper is implemented for each data store to pass data in either a relational representation or an XML representation, whichever is more appropriate for that data store. The system would also perform any necessary transformation and merging of data to support either an XML view or an SQL view, depending on the API called. (See Figure 5.)

These APIs would handle structured and semi-structured data quite well. However, they are not as suitable for unstructured data, especially for large-size, nontext content entities. First of all, such entities, in a large volume, often require a specialized server to manage, e.g., to provide hierarchical storage management (HSM) support and a policy-driven storage administration. Second, because of the sizes involved, direct delivery of these entities from the server to an application (and vice versa) is needed even in a multi-tier environment. Third, depending on the data type, some type-specific behavior (methods and constraints) may need to be maintained, or a special protocol may be needed to deliver the data. This leads to a type-specific server, and possibly even a specialized network connection. For example, streaming video to an application requires a large bandwidth, a push model, and a "quality of service" protocol. For these reasons, separate interfaces for unstructured data are needed. We propose using a URL (uniform resource locator) to identify each unstructured entity, including the hosting server and the protocol to use for accessing the entity. This URL is obtainable through either the XML APIs or the SQL APIs.

Challenges for information integration

We now discuss some of the major challenges the research community and the industry face in bringing together these different paradigms.

API, query language, and data model. As discussed in the last section, we believe there is a need for declarative query language support both for regular relational data (SQL) as well as for dealing with hierarchical data *a la* XML (XQuery). The challenge lies, however, in the ability to provide for higher-level semantics required by the CMSs either through extensions to the base languages or through new APIs. An example of a higher-level semantics construct would be providing for versioning of documents, schema, collections, and so on, with detailed change management. A good but primitive first step in this direction for document management systems is the WebDAV API.[23] Also, standards for meta-data are needed to describe the content,[24] and model collections of entities (e.g., "folders") and inter-entity relationships.

An integrated system also needs an ability to insert, update, and delete complex objects through the query languages or APIs supported. Specifically, syntax and semantics for XML modification operators (insert, update, and delete operations on a document) need further definition.

Transactional semantics. Although we did not discuss core systems issues such as concurrency control, recovery, and transaction models, we believe typical DBMS algorithms for addressing these problems do not apply as is for typical CMS transaction notions. Some requirements are an ability to support long duration locks for checkin/checkout capability, versioning, and stateless clients. Indeed, to support logging and recovery for an unstructured data repository such as Lotus Notes required substantive modifications to base ARIES-style recovery algorithms.[25]

In the context of providing update support across federated data sources, a multisite transaction is needed, requiring a two-phase commit[26] or equivalent protocol. This transaction is a challenge since not all the data stores necessarily offer such a capability. This inter-site synchronization problem goes beyond transaction support. It also requires a coordinated backup and recovery of the data stores participating in multisite transactions so that they can be restored to a mutually consistent state after a failure.

Next generation data federation. DBMS-based federated query processing needs to evolve to accommodate asynchronous, as well as slow, responses from a data store. Also, algorithms for providing

top-N ranked results for distributed data sources are going to become increasingly important.[27,28]

The system must accommodate large result sets. Cursor support is needed to allow an application to fetch only the portion of results that is needed. The challenge is how to provide an efficient cursor support when it is not supported by a data store, and how to support effective stream-based processing of rich media objects, XML documents and fragments, and collections of these entities.

Performance challenges. Rich media and semi-structured information (e.g., Hypertext Markup Language, or HTML, XML, etc.) increase the data volume requirements of the system from the terabyte (10^{12}) or petabyte (10^{15}) range to the exabyte (10^{18}) range. In this context, processing techniques to optimize for the first N responses will be crucial. Another interesting technical challenge is in the area of defining benchmarks to measure performance for CMS as well as enhancing benchmarks for DBMS to incorporate the information integration requirements (e.g., data federation, time to first response, and interactive querying).

XQuery is a new language and, as such, considerable work on query optimization and processing techniques is required. Perhaps, much like relational algebra and query processing algorithms,[29] XML query processing would benefit from a formal treatment of XQuery operations.

*Trademark or registered trademark of International Business Machines Corporation.

**Trademark or registered trademark of Transaction Processing Performance Council.

Cited references and notes

1. DB2 Product Family, IBM Corporation, see http://www.software.ibm.com/data/db2/.
2. IBM Content Manager portfolio, IBM Corporation, at http://www-3.ibm.com/software/data/cm/.
3. In this paper, heterogeneity refers to different types of data, not to schematic heterogeneity, which is an area of active research.[4,5]
4. R. Miller and L. Haas, principal investigators, and R. Fagin, M. Hernandez, L. Popa, Y. Velegrakis, X. Wang, and L.-L. Yan, members, of Project Clio: Managing Schematic Heterogeneity in Database Management Systems, see http://www.cs.toronto.edu/~miller/Research/hetero.html.
5. B. D. Czejdo, M. Rusinkiewicz, and D. W. Embley, "An Approach to Schema Integration and Query Formulation in Federated Database Systems," *Proceedings of the Third International Conference on Data Engineering* (ICDE) (February 3–5, 1987), pp. 477–484.
6. Nevertheless, we do believe a CMS ought to exploit RDBMS capabilities, including object-relational technology when it is suitable.
7. Some are RDBMS-based, whereas others are content-oriented.
8. CrossAccess Corporation, Data Integration Infrastructure, at http://www.crossaccess.com.
9. J. Kleewein, "Practical Issues with Commercial Use of Federated Databases," *Proceedings of the 22nd International Conference on Very Large Data Bases* (VLDB96), Bombay, India (September 1996), p. 580.
10. The Garlic Project, IBM Corporation, Research Division, http://www.almaden.ibm.com/cs/garlic/.
11. A. Tomasic, L. Raschid, and P. Valduriez, "Scaling Access to Distributed Heterogeneous Data Sources with DISCO," *IEEE Transactions on Knowledge and Data Engineering* **10**, No. 5, 808–823 (1998).
12. Y. Papakonstantinou, H. Garcia-Molina, and J. Wido, "Object Exchange Across Heterogeneous Information Sources," *IEEE 11th International Conference on Data Engineering* (ICDE) (March 1995), pp. 251–260.
13. L. M. Haas, D. Kossman, E. L. Wimmers, and J. Yang, "Optimizing Queries Across Diverse Data Sources," *Proceedings of the 23rd International Conference on Very Large Data Bases* (VLDB97) (1997), pp. 276–285.
14. Enterprise Information Portal, IBM Corporation, http://www.software.ibm.com/data/eip/.
15. M. A. Roth, D. C. Wolfson, J. C. Kleewein, and C. J. Nelin, "Information Integration: A New Generation of Information Technology," *IBM Systems Journal* **41**, No. 4, 563–577 (this issue, 2002).
16. D. Kossman, "The State of the Art in Distributed Query Processing," *ACM Computing Surveys* **32**, No. 4, 422–469 (December 2000).
17. R. Williams et al., *R*: An Overview of the Architecture*, Research Report RJ3325, IBM Corporation, San Jose, CA (December 1981).
18. E. Codd, "A Relational Model of Data for Large Shared Data Banks," *Communications of the ACM* **13**, No. 6, 377–387 (June 1970).
19. In contrast, in a DBMS, the schema definitions are sometimes considered the "meta-data," but it is not the same as CMS meta-data.
20. Here, "document" is merely an abstraction for an information entity that has a richer structural representation than a "record."
21. G. Wiederhold, "Intelligent Integration of Information," *Proceedings of the ACM Conference on Management of Data* (1993), pp. 434–437.
22. XML, XML Schema, XQuery, Xpath, see http://www.w3c.org.
23. Web-based Distributed Authoring and Versioning, see http://www.webdav.org.
24. Dublin Core Metadata Initiative, http://www.dublincore.org/.
25. C. Mohan, R. Barber, S. Watts, A. Somani, and M. Zaharioudakis, "Evolution of Groupware for Business Applications: A Database Perspective on Lotus Domino/Notes," *Proceedings of the 26th International Conference on Very Large Databases* (VLDB 2000), Cairo (September 2000), pp. 684–687.
26. J. Gray and A. Reuter, *Transaction Processing: Concepts and Techniques*, Morgan Kaufmann Publishers, San Francisco, CA (1993).
27. R. Fagin, "Combining Fuzzy Information from Multiple Systems," *Proceedings of ACM Symposium on Principles of Database Systems* (1996), pp. 216–226.
28. S. Chaudhari and L. Gravano, "Optimizing Queries over Multimedia Repositories," *Proceedings of the 1996 ACM*

International Conference on Management of Data (SIGMOD 96) (1996), pp. 91–102.
29. P. Selinger, M. Astrahan, D. Chamberlin, R. Lorie, and T. Price, "Access Path Selection in a Relational Database Management System," *Proceedings of the 1979 ACM SIGMOD International Conference on Management of Data*, Boston (1979), pp. 23–34.

Accepted for publication June 20, 2002.

Amit Somani *IBM Software Group, 555 Bailey Avenue, San Jose, California 95141 (electronic mail: asomani@us.ibm.com).* Mr. Somani is a senior software engineer and leads the unstructured information integration effort in the Data Management group. Prior to this, he was at IBM Research where he made significant contributions in the areas of data-driven e-commerce applications, log-based recovery technology for Lotus Notes, SMP-Query Parallelism for DB2 UDB, and TPC-C/TPC-D benchmarking for the IBM DB2/NT/PCServer. He has a B. Tech. in computer science and engineering from IT-BHU, India, and an M.S. in computer science from the University of Wisconsin at Madison.

David Choy *IBM Software Group, 555 Bailey Avenue, San Jose, California 95141 (electronic mail: dchoy@us.ibm.com).* Dr. Choy is the manager of the Database Technology Institute for Content Management at the IBM Silicon Valley Laboratory, responsible for content management and information integration technologies. He is the designer for the new IBM Content Manager (Version 8) architecture. Prior to joining the Silicon Valley Laboratory in 2000, he spent 26 years at the IBM Almaden Research Center, where he did research in relational DBMS, office systems, storage systems, and digital libraries. His professional activities included serving on the Executive Committee of the IEEE Computer Society Technical Activities Board for a number of years, and he authored the X/Open backup/recovery standard (XBSA). He holds a Ph.D. in electrical engineering from the University of Illinois at Urbana-Champaign.

James C. Kleewein *IBM Software Group, Silicon Valley Laboratory, 555 Bailey Avenue, San Jose, California 95141 (electronic mail: kleewein@us.ibm.com).* Mr. Kleewein is a Distinguished Engineer working in database architecture, strategy, and technology at the IBM Silicon Valley Laboratory. He has worked for IBM in the database business for the last 15 years. His technical expertise can be seen across a wide range of IBM data management products, including IMS, DB2/MVS, DB2/390, DB2 sysplex data sharing, DB2 Spatial Extender, DataJoiner, and DiscoveryLink™. Mr. Kleewein is a lead architect for Xperanto, focusing on expanding the role of DB2 from a structured data store to a structured and semistructured data store by adding XML capabilities to the DB2 engine.