

Managing Multimedia Assets with Federated Join

Alan Cole*, Jim Christensen*, Howard Sachar*, Oleg Dulin†

*IBM T.J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

†Clarkson University
P.O. Box 6100
Potsdam, NY 13699

E-mail: {colea, ibmjim, hsachar}@us.ibm.com, dulino@acm.org

Abstract

We describe the architecture of a federated system designed to allow the integration of structured, relational, data together with unstructured multimedia data. Two characteristics of multimedia have special implications for such an architecture. First, multimedia objects can be very large. Second, most good tools for working with multimedia expect these objects to be stored as files within a file system. These characteristics suggest that moving multimedia objects out of the file system, or making multiple copies, or moving them unnecessarily around a network, will all be suboptimal solutions. With these characteristics in mind, we experiment with a system architecture for federation that allows multimedia assets to remain in place, while still providing some of the benefits of structured data management.

1 Introduction

Many enterprises face the challenge of managing digital multimedia assets that exist as a combination of unstructured and structured data. The multimedia objects themselves, including still image, text, video, and audio, often reside as multimedia files in various file systems throughout the enterprise, both because tools to capture digital multimedia typically produce ordinary files, and tools to process or edit multimedia also expect ordinary files as input. Metadata describing or referencing these assets, however, are often stored in databases because of their highly refined support for queries, integrity, backup, and scalability. Because the unstructured data and structured data are at best loosely coupled, managing the combination is often unwieldy. Moving multimedia objects into a digital asset management system can make it difficult to use existing tools; keeping two copies, one in the file system and the other in the repository doubles storage requirements while introducing significant synchronization challenges.

This set of circumstances has motivated the experimental work reported here. We have developed an architecture in which we take the approach of leaving the digital media assets *in situ* and of providing them with a relational wrapper, and then of using federated joins to present the combination of unstructured and relational structured data as if they were all contained within a single relational table.

1.1 Current Challenges

The research reported on here was motivated by observations at a number of companies (including our own) where significant amounts of digital multimedia assets are created, acquired, used, and archived.

These multimedia assets consist of both unstructured multimedia objects and of structured metadata. For example, a video file of a television advertisement, which we can consider to be an unstructured object, probably has associated structured data giving details about the process of creating it (job tickets, project numbers, costs, dates, approvals), and of using it (markets, times, and costs for airing the advertisement).

Two general characteristics of multimedia objects heavily influence current and future architectures for incorporating them into an enterprise's information processing infrastructure.

The first characteristic is that multimedia objects are typically large, sometimes *very* large. This fact discourages any scheme in which multiple copies of objects are stored, or in which the objects are sent repeatedly over a network.

A second characteristic is that digital multimedia objects are most often stored as ordinary files within an operating system's file system. A substantial number of excellent tools have been developed to create, view, and edit various types of multimedia objects, and by far the majority of these tools expect their input to be a file, and produce their output as a file.

So, while the structured metadata may be stored in a database in order to take advantage of the query capabilities provided by database systems, as well as their support for integrity, reliability, and scalability, there is a big incentive to keep the multimedia objects as ordinary files. If the multimedia object is stored in the database, it becomes difficult for tools to access it; the object will have to be "checked out" before use, then checked back in after modification. If the object is stored in the database, but a copy is left in the file system for tool convenience, then the amount of storage required is immediately doubled and, in addition, it becomes difficult to keep the two sets of copies synchronized. Neither alternative is attractive.

Another observation is that both structured and unstructured data tend to be segmented along both departmental and geographic lines. Over time, "data silos" grow within the enterprise in response to immediate needs of the department or geography. Although some might view such developments as bad, departmentalization of data management and processing functions has many beneficial aspects. Tools, data storage location, and local control can all help to provide optimal solutions to the data processing needs of the individual department or area. For example, a company's European and North American offices will likely store much of their operational data locally, regardless of where the company headquarters are located, because this provides more efficient access to the needed data. This is especially true with multimedia data. The huge volume of this data dictates that it be kept as near as possible to its place of use, because the network bandwidth necessary to move these objects around is often not available.

As enterprises try to take the next step toward more efficient operation, however, there is an increasing motivation to effectively integrate and streamline the operations of individual departments or geographies. The approach we advocate here is that any such integration must be accomplished without destroying the current, optimized, departmental or geographic solutions. Any "solution" that requires replacing existing tools and processes with some grand new infrastructure is likely to have a severe and negative impact on day-to-day operations of existing departments, even if it does facilitate the development of new enterprise-wide applications.

2 Architecture and Implementation

Motivated by the belief that new enterprise-wide applications must not destroy or disrupt existing solutions that have been developed in response to the needs of individual departments or geographies, we have experimented with an approach that provides integration through federation of existing information sources, and which allows the existing information sources to continue to be used independently.

Two specific projects have provided the background to this work. The first was within the advertising industry, where a large and growing collection of digital multimedia assets, including video and still images, all represented advertisements for television or print publications, and were stored as ordinary files in a computer file system. Separately, a relational database was updated with information about advertisements, including when and where they had been aired or printed, estimated costs, company name, and so on. The database was used to generate various types of summary reports, and the multimedia files were used to review the actual content of the advertisements. There was no effective integration between the two information sources; users had to navigate manually between viewing the reports and viewing the multimedia objects.

A second project involved a database of products to be sold over the web, together with digital photographs of the products. To create the web site, these two separate sources of information had to be correlated and integrated, an operation which originally was performed manually.

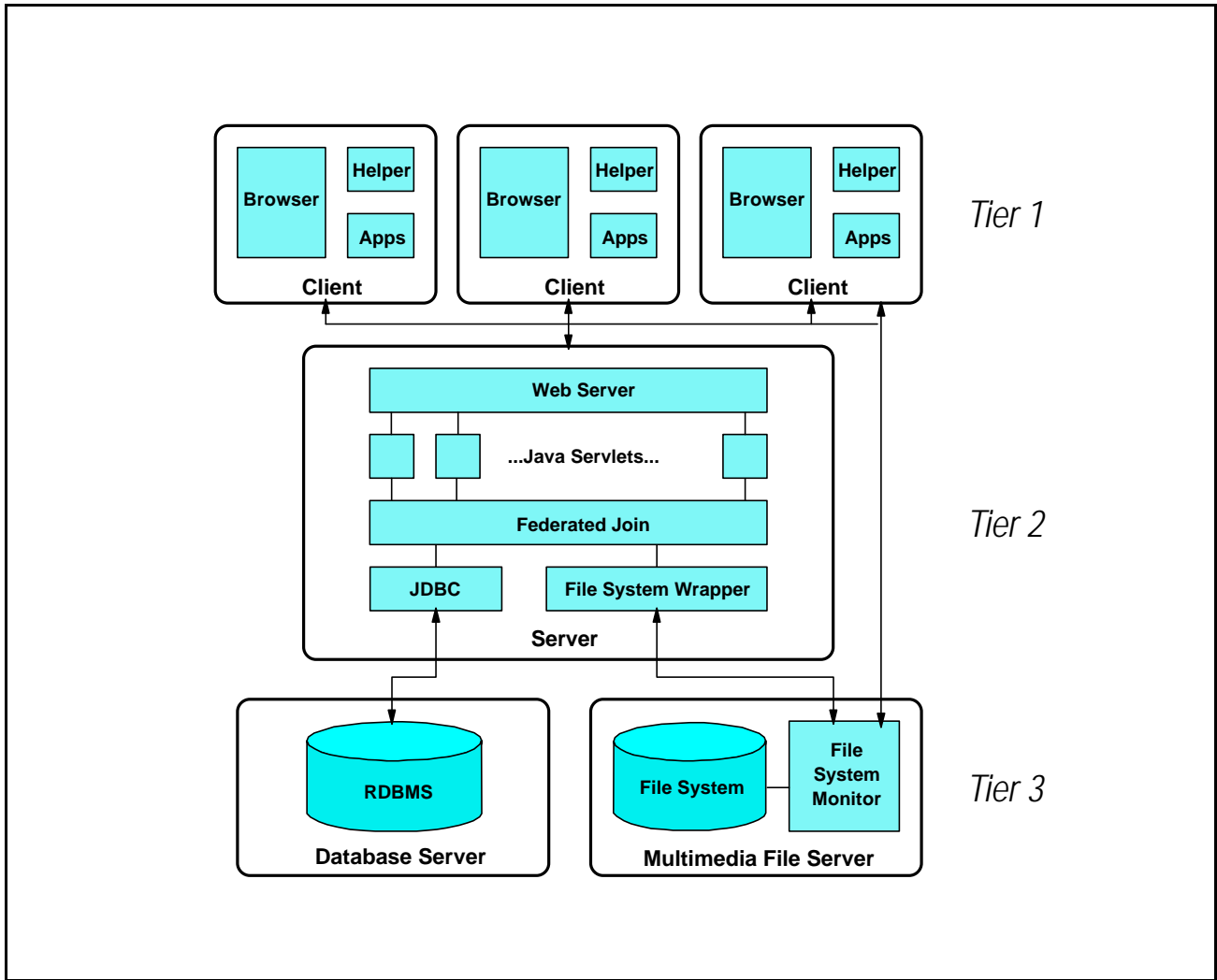


Figure 1: System Architecture

In approaching these projects, we have developed the three-tier architecture shown in Figure 1, and explained in greater detail in succeeding sections.

2.1 Tier 3: The Data Store

For both projects, two types of data sources are incorporated. The first type is an RDBMS. In the advertising case, the database contains information describing where and when the multimedia files were used. In the e-commerce case, the database contains information about the products for sale. In either case, the data model is straightforward to incorporate into our federated relational model, and existing vendor DBMS software is all that is required.

The second type of data source within both projects is a traditional file system containing multimedia files. Each such data source is provided with a wrapper (which actually exists on the tier 2 server) to provide a relational abstraction. A collection of multimedia files is considered to be a table, which each file corresponding to a row of the table. Specific columns depend on the type of multimedia. For a still image, for example, data include not

only a URL referencing the file itself, but also metadata such as the image dimensions, color depth, and other attributes extracted in real time from the individual file headers (e.g., a product identifier contained in an annotation tag). Attributes for other types of media are similar in concept, though different in exact details. For the projects discussed here, each multimedia file also has an associated “thumbnail” or proxy, typically a small-scale version of a still image, a key frame from a video, or a small version of a key image extracted from a print document. This thumbnail image is computed and updated as described below.

In addition to the relational wrapper, the tier 3 file system data store machine includes a File System Monitor program, which acts as a simple, dual-ported server.

On the first port, the File System Monitor exchanges information with the relational file system wrapper on tier 2, using XML-based descriptions of the media objects. The File System Monitor watches the collection of multimedia files and takes appropriate action when the collection changes. For example, if a new file is added to the collection, or an existing file updated, then the File System Monitor will create a new or updated thumbnail image. If a file is deleted, then the existing thumbnail image will also be deleted. In either case, the File System Monitor notifies the middle tier wrapper of any updates to the collection.

The File System Monitor also services a subset of HTTP requests on its second port. This allows clients to get multimedia files directly from the data store, and to store new or updated files in the collection, without having to go through the tier 2 server for these potentially huge files.

2.2 Tier 1: The Client

The user’s primary interface to the federation is via a web browser. Only one small, client-specific, program is required. This means that the effort required to support additional client platforms is minimal.

The client-specific program is a helper program that allows media objects to be edited and updated, even if they are stored on a remote data store. Rather than embedding direct references to media objects in the HTML sent to the client web browser, instead a reference to a small text file of a special MIME type is embedded. This file simply contains the URL of the media object. When the user clicks on a link in the browser, the small file is retrieved and the helper program, which has been associated with the special MIME type, is launched. The helper program reads the URL of the media object from its text file and then proceeds as outlined below.

The helper program has two responsibilities. One is to map URLs to ordinary file system names, if possible, using a configuration file. This is done for performance reasons. For example, the URL “http://9.2.147.53/media” might be mapped to the file system path “P:\PROJECT\IMAGES,” where “P:” is a remote drive accessible over the LAN (assuming a Windows file system for this example). In this case, the media URL “http://9.2.147.53/media/logo.tif” would be translated to the file system name “p:\PROJECT\IMAGES\logo.tif”.

If the URL for the media object does not translate into a file system name, then the helper program issues an HTTP **GET** request for the media object, which is stored in a local temporary file. Notice that this request is serviced by the file system monitor program on the tier 3 data store, bypassing the tier 2 machine entirely. This optimization is especially important when dealing with multimedia assets, to reduce overall network traffic.

The second responsibility of the helper program is to launch the appropriate media application and, optionally, to monitor it. This the program does by invoking whatever program is associated with the MIME type of the media object, passing the file system name of the object, either the name of the original object if the URL was mapped to a file system name, or the name of the temporary copy of the object if the URL could not be mapped to a file system name. In the above example, the program associated with image files in the TIFF format might be Adobe Photoshop, say, and so Photoshop would be launched on this image file. On another client machine, a different program might be associated with TIFF files, so on that machine, that program would be the one launched.

In the case where the URL has been mapped to a file name, the job of the helper program is now done. The application (e.g., Photoshop) can be used to edit the media object, and when the object is saved, the File System Monitor on the tier 3 data store machine will notice that the file has been updated, and will create an updated thumbnail image and update the tier 2 file system wrapper records.

In the second case, when the URL of the media object cannot be mapped to a file system name, the helper program monitors the temporary file. If the user edits the multimedia object and then saves an updated file from the application, the helper program will notice the update and perform an HTTP **PUT** operation to copy the media file back to the tier 3 data store. Once stored there, the File System Monitor will again observe the update and take appropriate actions (rebuild the thumbnail image and inform the tier 2 file system wrapper of the update).

The helper program is required on each client computer where it is desired to use the web interface, but still update media objects with native tools. For our current prototype, we have developed versions for the Microsoft Windows and Apple Macintosh operating systems.

2.3 Tier 2: The Server

The tier 2 layer acts as the integration layer and provides web services to the tier 1 clients. Clients interact directly with the web server, which can be any standard web server. Much of the content presented to the end-user is dynamically generated; this content is generated by Java servlets (executed by any one of a large number of available servlet engines). The abstract notion of federated tables, as well as organizing concepts such as notebooks and folders, and administration tools such as user authentication and table definition, are all represented by Java objects in the prototype implementation. These objects use commercial JDBC packages to access relational databases, and the custom file system wrapper, described above, to access multimedia file systems. The presentation servlets use the methods of these objects to obtain the information to be displayed.

The file system wrapper implements cursor stability for the file system data store through the use of a generation marking mechanism explained in [DC99]. This ensures that query results remain stable during the lifetime of the cursor.

2.4 The Federated Join

The integration of the structured, relational, metadata and the unstructured media objects is accomplished by defining a relational table that joins the relational data with the relational abstraction presented by the file system wrapper.

Within our prototype, this definition, normally performed by an administrator, is accomplished by means of a graphical user interface, rather than by writing SQL statements. In effect, an inner join with a limited subset of **WHERE** conditions is supported. The presentation logic, implemented by Java servlets, allows the end-user to scroll forward or backward through rows of the result table, to perform queries on the table, or to edit information in the table. To enhance performance, join results are cached by the table objects.

We have found it very useful to provide as one attribute of the multimedia abstraction a small thumbnail image representing the media object and which acts as a link (in the HTML sense) to the original object. These proxy images are the ones that are automatically created or updated by the File System Monitor on the tier 3 multimedia data store, and are typically a small version of a still image, a key frame from a video, and either a small version or an extracted image from a print presentation. (An audio asset might be represented by a short sound clip, but we have not arrived at a good visual proxy for an audio segment.)

The result is that the end-user, in browsing through the result table from the join, will see both the textual information as well as thumbnail images representing each multimedia asset. By clicking on the multimedia asset from within the browser, the user launches whatever application he or she has associated with that type of

media, as described in “*Tier 1: The Client*” above. In the case of still images, for example, the user might have Adobe Photoshop associated with TIFF files; clicking on the thumbnail representing a TIFF media object would then bring up Photoshop. The user could view or edit the image in the normal way, using his or her normal tool. If the image is updated, then saving it will result in both the actual media object being updated, and also its thumbnail proxy being updated by the tier 3 File System Monitor, as described above.

Outer Join with Reference Counts

We have also implemented a type of federated join, which we term “left outer join with reference counts,” which we have found to be particularly useful in the multimedia projects in which we have been involved. When defining the federated table, the administrator may specify that counts are to be calculated for the join column. This is similar to the aggregation performed by the **GROUP BY** clause, and is similar to a left outer join in that counts of zero are included.

The servlet that presents such a table generates the count field (for counts greater than zero) as an HTML link. By clicking on this link, the user is taken to a fully expanded sub-table in which the referenced row is disaggregated into the separate rows that were grouped to obtain the count.

To show why this can be so useful, let us take the example in which an e-commerce company has an existing catalog of structured data describing items for sale on its web site, each with a unique id, or SKU number. Separately, they will acquire or produce photographs, drawings, or video clips showing the items for sale, each annotated with a SKU number. These two information sources are typically provided by separate departments within the company, on different schedules. Two questions are of immediate interest to the overall project supervisor in trying to set up the web site:

- 1) How many SKUs in the catalog have no images? How many have multiple images?
- 2) How many images have no corresponding SKUs?

We use a federated left outer join table between the catalog and the multimedia wrapper, with reference counts, to answer the first question, and a similar join, but with the tables in reverse order, to answer the second question.

By browsing through the federated left outer join table in which the join column of SKU number is represented by a reference count, the supervisor can readily see how many images are associated with each item in the catalog. A query for counts of zero returns a result set showing all SKUs that do not have associated media objects. A query for counts greater than one returns a result set showing all SKUs that have more than one associated media object; clicking on the count field for any SKU will show an expanded sub-table in which each thumbnail is shown in a separate row, normally making it immediately obvious whether having multiple media objects associated with the SKU is due to an error (caused, for example, by a typographical error when annotating an object), or to some other cause (such as multiple photographs of the same item). Similarly, joining the tables in the opposite order yields a view in which the reference count shows media objects that do not (yet) have an associated entry in the catalog, or which are associated with multiple items in the catalog (which might be due to an error in the catalog, or might be legitimate if a video showing several different items is annotated with multiple SKU numbers).

An example portion of a federated result table from an advertising environment is shown below. A relational database gives information about each time a television advertisement was shown during some period. Separately, MPEG-format versions of the advertisements were stored on a separate system. The information was not integrated; to see the video mentioned in a particular database record, the user was required to manually start a separate application and locate the correct video file.

We defined a new table based on a federated outer join with reference counts. This table is shown in the user’s web browser and, as illustrated below, now shows a thumbnail image in each row. Furthermore, repeat instances of the same advertisement running at different times or places have been aggregated and represented by a count

that gives the total number of times the advertisement has run. The user may click on the count to see an expanded table showing each instance of the advertisement. The user may also click on the small image to play the video of the advertisement.

A further feature is that the table definition may indicate one or more summation columns. In the example here, “US Dollars” is such a column. The figure of \$13,107 in the first row, for example, represents the sum of the 13 separate instances for this advertisement.



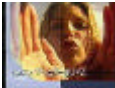

Filename	Caption	Icon	References	US Dollars
US376463	Services Expedition		<i>Count: 13</i>	13,107
US377270	Global Links		<i>Count: 16</i>	135,325
US377273	Sportswear Edition		<i>Count: 15</i>	49,617
US379009	Fun with tools		<i>Count: 26</i>	103,844

Table 1: Portion of a federated table with reference counts

3 Conclusions

Our construction of the prototype has been motivated by our past experiences in a number of projects involving multimedia objects (primarily still images). Information about the objects was typically structured and stored in a database. The multimedia objects themselves, because they had to be edited with tools that operated on ordinary file system files, had to be stored in the file system. Keeping the two sources of information synchronized was a largely manual process, tedious, error-prone, and labor-intensive.

The work described here grew out of these experiences. We have developed an approach that has the primary goal of being able to keep multimedia assets where they are needed to optimize use by current tools, but yet to allow them to participate in the more structured environment demanded by an integrated approach to workflow.

Initial experience with our prototype has been positive, and has convinced us that the architecture discussed here is a practical means for achieving efficient federation of large multimedia objects together with structured metadata. At the same time, existing applications which use the DBMS or which operate on the multimedia objects can continue to work as before.

4 Related Work and Future Directions

Earlier work on integrating distributed database systems (described in [SL90] and [LMR90], for example) has been important in influencing our approach to the problem, as has the more recent work on federating information sources which include semi-structured or unstructured data ([CHS99]). The work on the “Garlic” project at IBM’s Almaden Research Laboratory ([RS97]) has also influenced our approach.

One area in which further work is required is in incorporating better optimization techniques in federated joins and queries. Currently, we perform only the most rudimentary sort of optimization, and could benefit by incorporating better techniques, such as those described in [HKWY97] and [ROH99].

Although our experience with the architecture described here has been positive, we are also aware of some areas

in which further exploration of alternatives would be appropriate. For example, we considered writing a custom JDBC driver to interface with the tier 3 file system data store as an alternative to the file system wrapper with its private protocol. Our current approach was simpler, but the alternative would have offered benefits in providing a consistent interface to all the tier 3 data stores and might have made future extensions, such as better transaction support, easier to implement. We have also discussed evaluating the overall effect of placing the file system wrapper on the tier 3 machine rather than the tier 2 machine.

The performance of a system, and the ease with which it can be maintained, extended, and scaled to large problems, are complex functions of the many architectural decisions made in its design. One frustration that we have encountered is in instrumenting and measuring the performance of a complex, multi-user, distributed system. In our experience, performance measurements can vary wildly from one replication to the next, making it hard to know where system bottlenecks are located, and difficult to make objective comparisons of alternative architectures. We look forward to the introduction of better technologies in this area.

***Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries. Windows is a trademark or registered trademark of Microsoft Corporation. Adobe and Photoshop are trademarks or registered trademarks of Adobe Systems, Inc. Apple and Macintosh are trademarks of Apple Computer, Inc. registered in the US and other countries.*

References

- [CHS99] S. Conrad, W. Hasselbring and G. Saake, eds. *Engineering Federated Information Systems (Proc. EFIS '99)*. Infix-Verlag, Sankt Augustin, 1999.
- [DC99] O. Dulin and J. Christensen. Accessing Remote File Systems as Autonomous Information Sources. Available from the author dulino@acm.org, November 1999.
- [HKWY97] L. Haas, D. Kossmann, E. Wimmers and J. Yang. Optimizing Queries across Diverse Data Sources. *Proc. 23rd International Conference on Very Large Data Bases*, Athens, Greece, August 1997.
- [LMR90] W. Litwin, L. Mark and N. Roussopoulos. Interoperability of Multiple Autonomous Databases. *ACM Computing Surveys*, vol. 22, no. 3, September 1990.
- [ROH99] M. Roth, F. Özcan and L. Haas. Cost Models DO Matter: Providing Cost Information for Diverse Data Sources in a Federated System. IBM Technical Report RJ10141, San Jose, California, March 1999.
- [RS97] M. Roth and P. Schwarz. Don't Scrap It, Wrap It! An Architecture for Legacy Data Sources. *Proc. 23rd International Conference on Very Large Data Bases*, Athens, Greece, August 1997.
- [SL90] A. Sheth and J. Larson. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *ACM Computing Surveys*, vol. 22, no. 3, September 1990.