

Semantically Cohesive Metadata Fragments To Enhance JBI Information Retrieval

Felicia Harlow

Air Force Research Laboratory
2411 Avionics Circle
Wright-Patterson AFB, OH
45433, USA

felicia.harlow@wpafb.af.mil

Michael Talbert

Air Force Research Laboratory
2411 Avionics Circle
Wright-Patterson AFB, OH
45433, USA

michael.talbert@wpafb.af.mil

Robert Hillman

Air Force Research Laboratory
525 Brooks Road
Rome NY, 13441, USA

robert.hillman@rl.af.mil

[Approved for Public Release; distribution unlimited]

Abstract - *The Joint Battlespace Infosphere (JBI) was first introduced by the Air Force Scientific Advisory Board (SAB) in 1998 to realize the vision of a shared information space. The goal of this effort is to share a vast amount of information within and across communities of interest and allow aggregation, integration, fusion and dissemination of relevant (i.e., semantically related) information to enable more effective and timely decision making. This paper details an engineering methodology for the basic unit of data, the Information Object, that will introduce componentized IO type development within an infosphere. This enhancement will improve the ability of users to create and store Information Object (IO) type schemas, and query and subscribe to IOs, which may be semantically related by their inclusion of common metadata elements. Utilizing relational and Native XML database access methods, applying a component-based IO type development concept, and exploiting XML inclusion mechanisms, this methodology improves the means by which an information broker can deliver related IO types to subscribers from a single query or subscription. The proposal of this new IO type architecture also integrates IO type versioning, type coercion and namespacing standards into the methodology.*

Keywords: JBI (Joint Battlespace Infosphere), Native XML Databases, Information Object Retrieval, Metadata Fragments, XQuery

1. Introduction

In recent years, technology improvements have led to a dramatic increase in the amount of information available to military decision makers in the war-fighting arena. Interoperability of the systems which deliver this information has not seen such improvement, and consequently, the situational awareness for decision-making has not significantly improved. The challenge is the aggregation of all this data while delivering the appropriate level of information to users at all levels. The concept of a Joint Battlespace Infosphere (JBI) was introduced by the Air Force Scientific Advisory Board (SAB) and is in development within the Air Force Research Laboratory Information Directorate (AFRL) to realize the vision of a shared information space. In this document, they describe the vision of a combat information management system that that would integrate information from a wide variety of sources, aggregate that information, and then disseminate the information in the proper format and level of detail to appropriate consumers. In short, provide to the warfighter, subject to policy, access to the right information in the right format and at the right time [6].

Soon after the SAB published their reports, engineers, and sci-

entists within the Air Force Research Laboratory (AFRL/IF) began researching the technologies and techniques that one could exploit to solve this rather unique set of information management challenges. As a product of this research several in-house Reference Implementations (RIs) were developed in order to prove efficacy of approach while providing developers and researchers with a substrate upon which they could build their applications and research solutions. [4]. The JBI infrastructure can be viewed as a collection of information management services that will be used to maintain, disseminate, and otherwise manage information objects as they are published by producers and used by consumers. While significant progress has already been accomplished towards realizing the vision of the SAB, the research is still a work in progress. As such, there are many avenues to be explored to develop the optimum solution to fulfilling this information need.

This paper first examines a recent proposed database storage architecture modification and IO type schema conventions. After this case study is examined, recent and future improvements are discussed, which provided many enhancements to the performance and quality of information retrieval.

2. Background

The complexity of integrating a large volume of data requires more than a well-performing database storage technology. There is a requirement to assist users in locating all (and only) the data they need, possibly without knowing everything about the elements of metadata that describe that information. Put simply, there should be some semantic cohesiveness to the subsets of metadata elements that would allow users to obtain multiple different types of objects just by querying on these subsets of element(s).

First, it is necessary to explain the overall information architecture utilized within a JBI implementation. The basic unit of data is the Information Object (IO). Each IO type is composed of a set of metadata elements in an XML schema format. These metadata elements can be any valid XML elements except the base object metadata which is a fixed IO type, and integrated seamlessly into all new IO type schemas. A significant shortfall in the composition of IO types is that there is no common methodology, framework or standardization for structuring and defining new IO type schemas, and no way to perform a multiple-IO type search with a single query.

IO type schemas are defined and registered in a Metadata Repository (MSR), while the instances of these objects, and their respective payloads, are stored in an Information Object Repository (IOR). At the

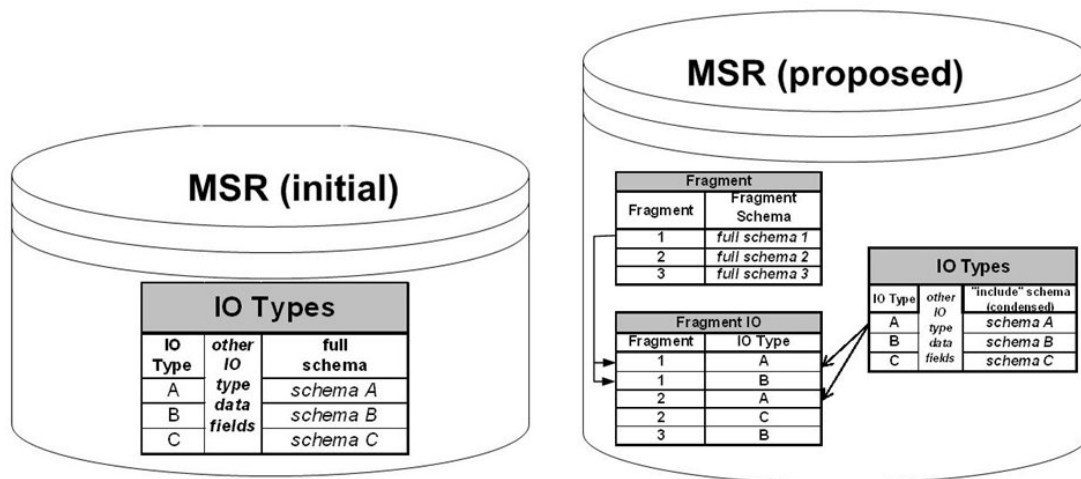


Figure 1: IO Types A, B & C which are comprised of fragments 1, 2 & 3 in Proposed MSR

time of this research (through mid-2005), the Reference Implementation (RI) utilized a relational database server (MySQL) for both the MSR and IOR. The MSR was composed of one database, containing one table, and one row in that table containing information about each IO type and its full metadata schema. The IOR storage and retrieval process in this relational database system is complex. Each IO type has its own table for storing the extent of IOs of that type. Within that table, each relative path of a node in the XML metadata tree is hashed to a unique numerical value using an XPath-to-SQL-92 conversion tool. This allows the entire object schema to be stored in a “flattened” database format by node. The payload is stored as an untyped BLOB (Binary Large Object). The IOR database contained one table for each IO type to hold the IO instances and a pointer to their respective payloads. The payloads of all IOs were stored in a separate common location. As one can surmise, scalability was limited because the number of IOR database tables and the collection of payloads could grow without bound and ultimately degrade performance.

The XPath-to-SQL-92 conversion tool also allows an XPath expression to be converted to a SQL expression for searching on these stored values. This process has a drawback in that repeated metadata paths within a metadata tree are not supported because they will hash to the same value. Consequently, the IO will retain all the distinct values stored at that repeated path, but only the last hashed/stored value will be available for predicate matching in the table. At the onset of this research, other approaches were being explored to alleviate this problem by utilizing a Native XML database. The recent implementation has adopted this solution and has provided even greater improvement opportunities (see Section 6 for implementation). For the purposes of having continuity within the context of this initial problem study, only the current method was used in this initial effort.

Another limitation is that each query or subscription can only search over one data type at a time. Thus, an application which required multiple IO type selections would have to submit one query or subscription call (each containing multiple steps) for each IO type.

3. Proposed Architecture

A previous research effort by one of the authors [5] proposed an enhancement of the database environment through the introduction of componentized IO type development. The IO types would be comprised of semantically-cohesive units of metadata, called *fragments*. This would show relations between IO types with common fragments

and allow for the introduction of multiple IO type searches. This enhancement was tested in the reference implementation. A simple conceptualization of the implementation of the initial and modified MSR is in Figure 1.

In this proposal, IO type schemas utilize XML inclusion techniques for building larger schema documents from smaller modular components. The XML Schema Part 0: Primer Second Edition W3C Recommendation (28 October 2004) has a language specifically available for including schemas (i.e., fragments) within other schemas [1]. The *include* element allows the reference to another schema within the same target namespace as the main schema. The *import* element allows the inclusions of these schema components from any namespace. In addition to modular schema development, the other improvements proposed and/or realized by this new IO type framework were:

- More effective use of database storage space in the Metadata Schema Repository (MSR) – through the use of a fragment table, a fragment-IO type pairing table (multiple IO Type entries for as many fragments it contains) and condensed IO type metadata schemas. The smaller schemas would use XML schema “include” statements and fragment declarations inside the root element, which are essentially pointers to an external fragment definition schema.
- Reduction in the time needed to build schemas, and subscribe to and/or query for objects – through fragment re-use and multiple IO type searches.
- Elimination of overhead associated with introducing revisions to IO type schemas – only fragments would be updated with newer versions and all that would be necessary for the IO type update would be changing the version number or fragment name in the fragment-IO type pairing table.
- Definition of standards and methods for versioning and coercion – to translate older IO instances to newer versions.
- Reduced IO type knowledge required by users to do a more search across all objects – via semantic-cohesiveness and re-use of common fragment types.
- Promotion of the use of namespaces to allow standardization and reuse of fragment schema components across multiple platforms – by defining fragments which would likely be used on many platforms. For example, many IO types would require a geospatial fragment describing the identifying location of an IO instance (e.g., for payloads containing video streams, images, etc.) A central namespace would ensure a user could search across multiple platforms which adhere to some of the standard metadata fragment definitions.

A new MSR was created with the fragment solution (copied and modified from the old MSR). The new MSR contained 10 fragments and the IO types were composed of every possible combination of these fragments (1023). Each fragment combination was present in from 1 to 512 of the IO types, based on all possible combinations of the 10 fragments (i.e., 10 choose 1, 10 choose 2, etc.). A simple Java program was tested outside of the context of the reference implementation.

The query method was slightly different for the original and the new MSR. The new MSR test consisted of predicate tests using only the fragment tables while the original MSR test was tested on the original schemas in the original MSR (as opposed to the newer condensed schemas in the new MSR). This was required because namespace support was not yet supported so XML includes could not be processed. Consequently, the original MSR IO type schemas would have to be used for the original MSR test. Given an array of fragments and a predicate expression containing fragment metadata, the test program was run for three different MSR scenarios:

1. Query using the new (fragment-based) MSR fragment.IO table to retrieve the matching IO types for the given fragment array. Store these matching IO types in a vector.
2. Execute a single SQL query to the original MSR to retrieve a large result set containing the name and schema every IO type in the table. Iterate through the result set to find the IO types which contain at least all the fragments in the predicate expression. This method trades overall SQL query execution time at the expense of greater memory consumption, and should theoretically take less time to execute than the next technique.
3. Iterate through the entire MSR with one SQL query per row. Retrieve and test each result to find the IO types which contain at least every fragment in the predicate expression. This method would consume less memory than the previous scenario, at the expense of greater overall SQL query execution time. It should have the greatest execution time.

4. Test Results

The results for the three scenarios are shown in Figure 2. The SQL queries were also run outside of the program, from the MySQL Control Center. This was done to isolate the SQL time from the rest of the program and determine the percentage of time attributable to iteration. The isolated SQL query times are shown in Table 1. For the current

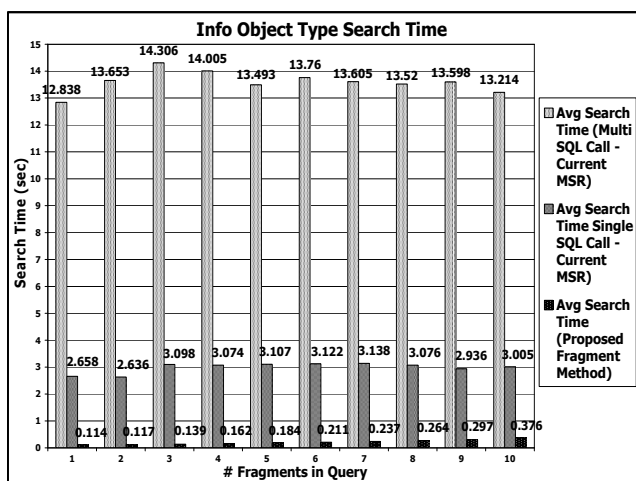


Figure 2: Info Object Type Search Time Comparison

MSR techniques, there was no change to the query at all, because the same single query or iteration of queries was the same for every test. As such, the results did not change and had nothing to do with the number of fragments in the predicate. These execution times should be high for the multiple SQL call test because of the number of IO type schemas (1023) which would have to be retrieved individually. Indeed, the longest execution times were for this configuration. Much improvement was realized by the single SQL query method, although approximately 90% of the execution time of this method was again attributable to the iteration through the IO type schema result set because only one retrieval was required. The proposed fragment method had the lowest execution time and most of that time was attributable only to the SQL query. The application cost only accounted for an average of 12% of the overall execution time. Using the proposed method results

Table 1: Isolated SQL Execution Times (sec)

Number of Fragments in Array/Predicate	AVG SQL TIME Proposed	AVG Multi SQL TIME Current MSR	AVG Single SQL TIME Current MSR
1	0.09	12.94	0.18
2	0.10	12.94	0.18
3	0.12	12.94	0.18
4	0.14	12.94	0.18
5	0.17	12.94	0.18
6	0.19	12.94	0.18
7	0.22	12.94	0.18
8	0.24	12.94	0.18
9	0.27	12.94	0.18
10	0.33	12.94	0.18

as a baseline, Table 2 shows the improvement of the fragment method over both of the current MSR options for the program scenarios. As hypothesized, the fragment solution provides improvement over the iterative search methods by at least a factor of 8. The execution time has increased as the number of fragments in a query increases. This corresponds to an increase in the number of arguments in the search predicate. This will be examined further in the analysis of these test results to see if the execution time approaches a limit. If it does, the *minimum* improvement can be stated. Each search was performed on a large number of samples and the execution times averaged for each scenario.

5. Performance Analysis

Although the testing demonstrated that a fragment-based MSR provided better execution time performance for predicate searches than the current architecture, the level of this improvement must be evaluated. The first question that should be considered is the decreasing level of improvement corresponding to a greater number of fragments searched for in the query.

In this isolated execution environment, the lengthier execution time cannot be attributed to an increase in the database size, which was constant for all tests. Therefore, the number of expressions in the search predicate and/or the aggregation required by grouping the results by IO type are the factors which must have affected the increase in response time. Of course, that is not to say that an increase in database size will not cause an increase in execution time, because database growth will certainly impact the execution time. The degree to which this growth will increase execution time is not predictable because the

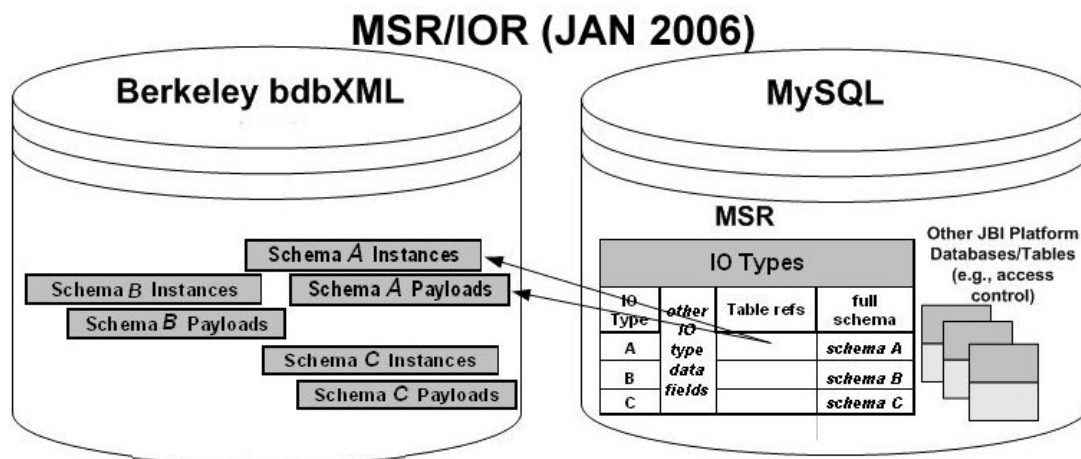


Figure 3: Current MSR/IOR

Table 2: IO Type Search Performance Improvement

NUMBER OF FRAGMENTS IN PREDICATE	PERFORMANCE IMPROVEMENT: Fragment Method vs. Current (Multi SQL Call)	PERFORMANCE IMPROVEMENT: Fragment Method vs. Current (Single SQL Call)
1	112.47	23.287
2	116.74	22.537
3	103.02	22.306
4	86.66	19.021
5	73.16	16.926
6	65.22	14.727
7	57.50	13.263
8	51.27	11.662
9	45.72	9.873
10	35.19	8.003

maximum database size cannot be approximated. However, in this testing database size was not a factor in the increase in execution time because the database size remained constant.

What must be considered is whether there is a predictable maximum number of fragments that may be needed to specify the IO type(s) of interest, and what is the cost to execute a search with this number of fragments in the predicate. This number should represent the maximum number of fragments contained in any particular IO type schema in an MSR. Otherwise, there would be no reason for a user to search for matching IO types with more than this number of fragments (as there could be no IO type with more fragments). The challenge is that there is no way to accurately predict this number. However, the structure of the information object is that only searchable metadata fragments should be present in the IO type schema (other elements can be part of the object payload). If an analogy can be made to other searching techniques (such as an internet search engine), an assumption can be made that the initial scope of the query (i.e., the number of fragments in the query) would be smaller than the maximum size to allow the broadest search and that additional fragments would be added only to scale down a large result set. As this is only extrapolation, another test was conducted to expand the query results and allow more confidence in the conclusion.

A test was done to examine the limits of expanding the fragment.io

search predicate. For this evaluation, 10 new test fragments were added to the fragment table and the fragment.io table size was doubled with the addition of new test fragment.IO type pairs. These new pairs contained existing IO types paired with the new fragments. The goal was to create a valid search predicate containing 20 elements arguments. This would provide results which could be used to evaluate whether the increase in table size or the search predicate would cause the SQL execution time to continue to increase or level out.

The testing was a series of queries with a 20 argument search predicate executed within the MySQL Control Center database environment. The average execution time for these queries was 300 milliseconds. Comparing this to the execution time for 10 fragments in Table 1, it is clear that the execution time does not continue to grow even though the number of elements *and* the table size were both doubled.

This test affirms that the fragment-based MSR greatly outperforms (by at least a factor of 8, as previously shown) the two iterative MSR searches with at least 1,000 IO type schemas containing a total of at least 10,000 entries within the fragment.io table. The fact that there was no change to the maximum execution time allows that there may be room for many more IO types and fragments with no performance degradation. The cost of the iterative search will *certainly* increase with the addition of IO types, making the fragment choice even more advantageous in that situation.

6. Recent & Future Improvements

The JBI RI version 1.2.6, which was released in January 2006, uses a hybrid approach to data storage and information retrieval. It still uses MySQL as the relational database server and has added a native XML database server, Berkeley dbXML (hereafter referred to as bdbxml), for a re-engineered MSR and IOR. The new architecture is shown in Figure 3.

There are several key elements to take note of in this new design. First, MySQL still maintains a full version of each IO type schema in the MSR since this makes it readily and quickly accessible for many tasks. In addition, this table still maintains additional IO type schema "bookkeeping" data (i.e., version number, description). More importantly, the MSR now contains pointers for each IO type to two bdbXML server databases that contain the IO instances and payloads of that type. This *directly* addresses the scalability issues of both the boundless growth of the number of tables in the IOR, and the

degradation of payload retrieval from a single collection of all IO type payloads.

The next major improvement pertains to using a native XML database such as bdbXML. First, there is no longer a need to do any XPath-to-SQL conversion to search a node value or path in the “flattened” relational database. Native XML databases provide very efficient document storage and retrieval, particularly because they are able to map a document by node (for this purpose, an IO type instance) for faster query execution.

The latest version of bdbxml has added support for the new XQuery language (still a working draft as of Feb 2006) being developed by the World Wide Web Consortium (W3C). This language is a more powerful extension of the XPath query language (version 2.0) and is designed to return the same results for a given query [3], but has added significant capabilities. Of major significance to the reference implementation is that XQuery can provide cross-document (thus, cross-IO type) searches on fragments in different collections (databases). As the developer’s (SleepyCat Software) website states in its key features manual “The XQuery language brings to XML databases what SQL brings to relational databases. With XQuery it is easy to express complex relationships, joins, conditions and result sets in statments that can be optimized and executed quickly over huge data sets” [2]. Finally, bdbxml provides storage retrieval of both XML and non-XML data within the same transaction, with no mapping or translation required. This is a particular feature needed for any object retrieval need that requires both the object (in whatever format), as well as the metadata to be returned. It allows a single transaction to retrieve the IO instance and its payload in whatever form.

7. Conclusion & Future Work

The initial research effort sought to provide a multi-object search capability and detailed some scalability issues and other objectives. The scalability problem has been alleviated and namespace support has been added via the introduction of bdbxml and the hybrid of data storage technologies.

The latest version of the JBI RI does not yet support the XQuery capabilities of bdbxml. However, it is planned for a future release and will allow for searches on semantically cohesive fragments. That was the primary objective of the initial research effort, the achievement of which the new database technology and XML language enhancements have made possible. The means now exist to accomplish the following: a) introduce a central namespace where some initial fragment definitions can be provided, and b) incorporate XQuery capabilities to allow for multi-object type searches.

8. Acknowledgments

This work was greatly supported and funded in part by the Air Force Research Laboratory Information Directorate’s Systems and Information Interoperability Branch.

9. References

- [1] *XML Schema Part 0: Primer Second Edition W3C Recommendation 28 October 2004*, May 2005. <http://www.w3.org/TR/xmlschema-0/>.
- [2] Berkeley DB XML 2.2 Overview, Feb 2006. <http://www.sleepycat.com/products/bdbxml.html>.
- [3] *XQuery 1.0 : An XML Query Language 3 November 2005*, Feb 2006. <http://www.w3.org/TR/xquery/>.
- [4] V. T. Combs, R. G. Hillman, M. T. Muccio, and R. W. McKeel. “Joint Battlespace Infosphere: Information Management within a C2 Enterprise”. In *The International Command and Control Technology Symposium*, Jun 2005.
- [5] F. Harlow. “a JBI Information Object Engineering Environment Utilizing Metadata Fragments for Refining Searches on Semantically Related Object Types”. Master’s thesis, Air Force Institute of Technology, Jun 2005.
- [6] USAF Scientific Advisory Board. *Building the Joint Battlespace Infosphere*, Dec 2000. SAB-TR-99-02.