Air Force Institute of Technology
AFIT Scholar

Theses and Dissertations Student Graduate Works

6-2005

A JBI Information Object Engineering Environment Utilizing Metadata Fragments for Refining Searches on Semantically-Related Object Types

Felicia N. Harlow

Follow this and additional works at: https://scholar.afit.edu/etd

Part of the Computer Engineering Commons

Recommended Citation

Harlow, Felicia N., "A JBI Information Object Engineering Environment Utilizing Metadata Fragments for Refining Searches on Semantically-Related Object Types" (2005). *Theses and Dissertations*. 3838.
https://scholar.afit.edu/etd/3838

This Thesis is brought to you for free and open access by the Student Graduate Works at AFIT Scholar. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of AFIT Scholar. For more information, please contact richard.mansfield@afit.edu.

المنارة للاستشارات

www.manaraa.com

A JBI Information Object Engineering Environment Utilizing Metadata Fragments for Refining Searches on Semantically-Related Object Types

THESIS

Felicia N. Harlow, Captain, USAF

AFIT/GCE/ENG/05-03

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

المنارة للاستشارات
www.manaraa.com

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the United States Government.

www.manaraa.com

AFIT/GCE/ENG/05-03

A JBI Information Object Engineering Environment Utilizing Metadata Fragments for Refining Searches on Semantically-Related Object Types

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science in Computer Engineering

Felicia N. Harlow, B.S.

Captain, USAF

June 2005

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

المنارة للاستشارات

www.manaraa.com

AFIT/GCE/ENG/05-03

A JBI Information Object Engineering Environment Utilizing Metadata Fragments for Refining Searches on Semantically-Related Object Types

Felicia N. Harlow, B.S.

Captain, USAF

Approved:

/signed/	2 June 2005
LtCol Michael L. Talbert (Chairman)	date
/signed/	2 June 2005
Maj Christopher B. Mayer (Member)	date
/signed/	2 June 2005
Dr. Kenneth M. Hopkinson (Member)	date

المنارة للاستشارات

www.manaraa.com

Acknowledgements

First and foremost I would like to thank my thesis advisor, Lt. Col. Michael L. Talbert, for his guidance, direction and feedback. His numerous constructive comments and our lengthy discussions have greatly improved my work.

I also owe a debt of gratitude to my sponsor, AFRL Information Directorate, and specifically Robert Hillman for the inspiration and financial support for my work. I am also very appreciative of the assistance and professionalism of the entire JBI inhouse team, who provided periodic feedback for my work and answered my numerous questions during this effort.

I'd also like to thank my readers, Maj Christopher B. Mayer and Dr. Kenneth M. Hopkinson, for their thoughtful contributions and insights.

I am extremely grateful to my husband for his incredible support and understanding during this research endeavor.

Finally, I would like to thank the rest of my family, for their encouragement throughout all of my academic pursuits and for always believing in me.

Felicia N. Harlow

المنارة للاستشارات
www.manaraa.com

Table of Contents

Page

Acknowledgements iv

List of Figures ix

List of Tables x

List of Abbreviations xi

Abstract xiii

I. Introduction 1
- 1.1 Background 2
- 1.2 Research Objectives 3
- 1.3 Assumptions 4
- 1.4 Approach 5
- 1.5 Research Sponsor 5
- 1.6 Summary 5

II. Background 7
- 2.1 Introduction 7
- 2.2 JBI 7
  - 2.2.1 Key Concepts 7
  - 2.2.2 Architecture of Reference Implementation . . . 10
  - 2.2.3 Information Objects (IOs) 10
  - 2.2.4 Common Application Programming Interface . . 18
  - 2.2.5 Information Dissemination 18
  - 2.2.6 Information Retrieval 18
- 2.3 XML 19
  - 2.3.1 XML Rules 19
  - 2.3.2 XML Schemas and DTDs 20
  - 2.3.3 XML Namespaces 21
  - 2.3.4 XPATH 25
  - 2.3.5 XML Inclusion Methods 26
  - 2.3.6 XML Validators 27
  - 2.3.7 Distributed Schema Design 27
- 2.4 DOD Metadata Registry 29
- 2.5 Summary 30

www.manaraa.com

Page

III. Methodology 32
3.1 Introduction 32
3.2 Problem Definition 32
3.3 Primary Objective: Common IO Engineering Framework 33
3.4 Improvements 35
3.4.1 Multi-Object Search 35
3.4.2 Better Schema Storage Method 36
3.4.3 Less Effort to Build, Subscribe, Query 36
3.4.4 Simpler Object Schema Revision Rules 37
3.4.5 Versioning and Coercion Methods 38
3.4.6 Less IO Type Knowledge Required by Clients . 39
3.4.7 Take Advantage of XML Namespaces 40
3.5 Solution Approach 40
3.5.1 Introduce A Component-Based Schema Structure 41
3.5.2 Fragment Naming Conventions 41
3.5.3 Fragment Schema Elements 43
3.5.4 Central Namespace for Fragments 44
3.5.5 XML Inclusion to Build Schemas 45
3.5.6 Fragment Tables 45
3.5.7 Fragment Processing Techniques 48
3.5.8 Versioning Standards 49
3.5.9 Changes to Pub/Sub/Query 50
3.5.10 Coercion Techniques 52
3.5.11 Information Object Storage Modifications . . . 56
3.5.12 Common API Changes 57
3.6 Environmental Parameters 58
3.7 Evaluation 59
3.8 Hypothesis and Interpretation of Results 59
3.9 Summary 59

IV. Implementation 61
4.1 Introduction 61
4.2 Compromises 61
4.3 Preliminary Operations 62
4.3.1 Fragment Library 62
4.3.2 Platform Limitations 64
4.3.3 Populating Database Tables 65
4.3.4 Permissions 65
4.3.5 IOR Table Fields 65

المنارة للاستشارات
www.manaraa.com

Page

4.3.6 Combination Generator 66
4.3.7 Creating Fragment Schemas and Instances . . . 67
4.3.8 Creating and Populating Files and Tables . . . 68
4.4 Test Application 68
4.4.1 Evaluation Parameters 69
4.4.2 Current MSR Evaluation Modifications 70
4.4.3 Current MSR IO Type Search 71
4.4.4 XML Instance to Schema Validator 72
4.4.5 Proposed MSR IO Type Search 72
4.4.6 Application Interface 72
4.4.7 Evaluation Measurements 73
4.5 Summary 76

V. Results and Evaluation 77
5.1 Introduction 77
5.2 Testing 77
5.2.1 Testing Environment 77
5.2.2 Evaluation Approach and Assumptions 78
5.2.3 Retrieval Options 78
5.2.4 Limitations and Validity 79
5.2.5 Testing Procedure 80
5.2.6 Test Results 81
5.3 Observations and Analysis 84
5.3.1 Search Time Improvement 85
5.3.2 Schema Revision Efficiency 87
5.3.3 Schema Storage Improvements 87
5.3.4 Reduced IO Type Knowledge Requirements . . 88
5.3.5 Versioning, Coercion and Namespaces 88
5.3.6 Integration Issues 89
5.3.7 Shortfalls and Compromises 91
5.4 Summary 92

VI. Conclusion and Future Work 93
6.1 Introduction 93
6.2 Main Research Contributions 93
6.3 Future Work 93
6.3.1 Common API Expansion 93
6.3.2 Fragment Library 94
6.3.3 Platform Recommendations 94
6.3.4 Additional Exploration 95
6.4 Summary of Research 96

المنارة للاستشارات
www.manaraa.com

Page

Appendix A. Fragment Data 97

Appendix B. Source Code 98
B.1 Fragment Create Table Statements 98
B.2 Combination Generator 98
B.3 Current MSR IO Type Search 1 101
B.4 Current MSR IO Type Search 2 102
B.5 XPath Evaluator 103
B.6 Proposed MSR IO Type Retrieval 104

Bibliography 106

المنارة للاستشارات
www.manaraa.com

List of Figures

Figure		Page
2.1.	JBI Concepts [16]	8
2.2.	Base Information Object Metadata Elements [2]	12
2.3.	MSR Table	13
2.4.	MSR Schema Viewer	14
2.5.	Sample Schema Instance With Hash Map	15
2.6.	IOR Table	16
2.7.	Inheritance Using Extension [2]	17
2.8.	XML Basic Syntax	20
2.9.	Namespacing Methods [8]	24
2.10.	Restriction [12]	28
2.11.	Using xsd:any for Optional Schema Elements [12]	29
3.1.	Fragment Schema Files	42
3.2.	Component Schema with includes	46
3.3.	Fragment Tables	47
3.4.	Coercion Example (geospatial_frag: version 1.0 to 2.0)	55
4.1.	JBI Fragment Query Tester	73
5.1.	Info Object Type Search Time Comparison	84
5.2.	Fragment Query Process	90
A.1.	Fragment Data	97

المنارة للاستشارات

www.manaraa.com

List of Tables

Table Page

2.1. Namespacing Methods 22

3.1. New MSR Tables 48

3.2. Coercion Table Structure 53

3.3. Coercion Table Change Descriptions 56

4.1. Fragment Schema Distribution 64

4.2. Fragment Names 66

4.3. Preprocessing Programs 69

4.4. Cost Comparison of Methods 76

5.1. Number of Samples per Number of Fragments 82

5.2. SQL Execution Times (sec) 83

5.3. Performance Improvement 85

المنارة للاستشارات
www.manaraa.com

List of Abbreviations

Abbreviation		Page
JBI	Joint Battlespace Infosphere	1
SAB	Scientific Advisory Board	1
AFRL	Air Force Research Laboratory	1
COTS	commercial-off-the-shelf	2
JBI	Joint Battlespace Infosphere	7
IEIST	Insertion of Embedded Infosphere Support Technologies	9
GA	Guardian Agent	9
HA	Host Agent	9
RI	Reference Implementation	10
J2EE	Java 2 Enterprise Edition	10
RDBMS	Relational Database Management System	10
IOR	Information Object Repository	10
IOs	Information Objects	10
MSR	Metadata Schema Repository	11
COI	Community of Interest	11
CAPI	Common Application Programming Interface	18
IR	Information Retrieval	18
XML	Extensible Markup Language	19
SGML	Standard Generalized Markup Language	19
HTML	Hypertext Markup Language	19
DTD	Document Type Definition	20
W3C	World Wide Web Consortium	21
URI	Uniform Resource Identifier	21
DOD	Department of Defense	29
RDF	Resource Description Framework	29

المنارة للاستشارات
www.manaraa.com

Abbreviation		Page
OWL	Web Ontology Language	29
CMD	Common Mission Definition	30
SQL	Structured Query Language	57

المنارة للاستشارات

www.manaraa.com

AFIT/GCE/ENG/05-03

Abstract

The concept of a 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 the JBI concept is to interconnect a collection of rigid, stove-piped C2 systems to a shared information space which will perform aggregation, integration, fusion and dissemination of relevant (i.e., semantically related) battlespace information which will enable the most effective and timely decision making.

The JBI Reference Implementation (RI) developed by the Air Force Research Laboratory (AFRL) in-house team is a suite of core web services, persistence, security and client interface methods that is implementing the SAB vision incrementally by adding new key services with each version release. While great improvement has been realized with each successive release, the engineering of the the basic unit of data within a JBI, the information object (IO) has only been minimally addressed.

This research proposes an IO engineering methodology that will introduce componentized IO type development. This enhancement will improve the ability of JBI users to create and store IO type schemas, and query and subscribe to information objects, which may be semantically related by their inclusion of common metadata elements. Several parallel efforts are being explored to enable efficient storage and retrieval of IOs. Utilizing relational database access methods, applying a component-based IO type development concept, and exploiting XML inclusion mechanisms, this research improves the means by which a JBI 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. The combined proposed framework provides a better means by which a JBI can deliver the right information to the right users at the right time.

www.manaraa.com

A JBI Information Object Engineering Environment Utilizing Metadata Fragments for Refining Searches on Semantically-Related Object Types

I. 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 improved. Insufficient information has been replaced with information overload. The new challenge is the aggregation of all this data while delivering the appropriate level of information to users at all levels. Getting the right information to the right people at the right time will provide a rich "information landscape" that will ensure information dominance in future engagements. The concept of a Joint Battlespace Infosphere (JBI) was introduced by the Air Force Scientific Advisory Board (SAB) in two technical reports: "Information Management to Support the Warrior" [1] and "Building the Joint Battlespace Infosphere" [26]. A team was formed within the Air Force Research Laboratory Information Directorate (AFRL) to realize the vision of a shared information space. The ultimate goal of the JBI concept is to interconnect a collection of rigid, stove-piped systems to a shared information space which will perform aggregation, integration, fusion and dissemination of relevant battlespace information which will enable the most effective decision making. Even though much development has already been accomplished towards realizing the vision of the SAB, the JBI concept is still a work in progress. As such, there are many issues that have yet to be explored to develop the optimum solution to fulfilling this information need. The complexity of integrating such a large volume of data requires that some method of indexing, cataloging and/or referencing be employed to assist the end user

المنارة للاستشارات
www.manaraa.com

in locating their pertinent information. Those issues are what prompted this research. The goal of this research is to explore the repository storage and object retrieval mechanisms within the JBI to determine whether improvements can be made to the services provided by a JBI.

1.1 Background

The JBI is a combat information management framework that provides individual users with the specific information required for their functional responsibilities during crisis or conflict. The JBI integrates data from a wide variety of sources, aggregates this information, and distributes the information in the appropriate form and level of detail to users at all echelons [26]. The previous two statements are a bit misleading, because they speak of "the" JBI, when in fact the concept described is "a" JBI instance. The JBI is not one central system that supports all operations. Rather, a JBI is established when deemed necessary, based on the development of a crisis or contingency. Of course, some ongoing operations will require a constant JBI. The JBI concept provides a standards-based open system and extensible infrastructure upon which legacy, evolving and future information systems will operate [19].

A JBI "Platform" consists of the set of core services which allow clients to store and retrieve information objects from a JBI data store. The platform contains an application server, data storage technology and client interface methods. The platform uses the subset of web services consisting of publish, subscribe and query to allow clients to store and retrieve objects of interest.

The Air Force has made a significant investment in its own science and technology sector while still taking advantage of the advancements made in the commercial sector and with commercial-off-the-shelf (COTS) technologies. The involvement of the Air Force science and technology community is required to ensure the many unique requirements of military operations are addressed. To spur rapid advancement, there have been several experimental JBI prototypes developed in past years. Each instance took a differing technical approach to development with a focus one or more of three

المنارة للاستشارات
www.manaraa.com

main areas: user connections through middleware, integration of legacy C2 systems and development of enabling science and technology.

1.2 Research Objectives

The objective of this research is to improve the quality of service provided by the JBI. It should be anticipated that once fully deployed, a particular instance of a JBI must be significantly scalable. It is easy to envision this need because today's typical military operation is a large scale operation of short duration. Therefore, a JBI instance can range from a small day-to-day State-side operational information exchange to supporting a large Air Operations Center coordinating a Joint Force operation over an entire region of the world.

It may be difficult to quantitatively measure the improvement realized from a system that is still in development and does not have a large scale deployed model on which to make an evaluation. However, this research focused on modifying the structure of the basic unit of measurement within a JBI, the Information Object (IO). That, in turn, will directly impact how easily a JBI delivers the right information to the right users because it introduces a new way to relate previously unrelated but potentially equally relevant information. What can be measured is how much relevant information is published that should be delivered to a user and how much of that relevant information the user received for a defined information need. This evaluation of the improvement achieved with the proposed solution must take into account how much "effort" was expended to retrieve this information.

The current effort expended by a user to retrieve data from a JBI platform through subscribe or query is finding the right information server and then finding and subscribing to the particular data objects of interest. Objects within a platform are of a specific *object type* as described by their *metadata schemas*. A single subscription is for objects of any single object type. The current level of service required to perform these operations requires that the user knows how to find and connect to the particular platform on which these objects are stored and that each object type of

www.manaraa.com

interest be subscribed to or queried individually. Simply stated, there are no current semantic correlations between object types nor is there a method of discovery of JBI platforms. Whereas the current JBI reference implementation only allows a search by a single object type, this research introduces an object type correlation that will instead allow users to do an expanded search to include multiple object types with common components, thereby allowing the user to search across all object types with these common components.

1.3 Assumptions

Several assumptions are required to allow a proposed improvement to be implemented and evaluated. Most of these assumptions are required due to the fact that the JBI implementation is still in development and undergoing significant changes with each new release. As such, there is no typical user and no typical platform parameters. It is in these areas that assumptions are made that may not accurately reflect what will become the common platform or user. These assumptions are:

- There is only one JBI platform over which a subscription or query can be executed. This assumption is to aid in evaluation and because there is currently no mechanism for platform discovery.
- The user modeled is not assumed to know the structure of every IO (this is to model a large IO environment, with too many objects to manage by a user with information needs). However, it is also assumed that there are certain metadata conventions within the platform for common elements and these will be available in a platform catalog to enable the user to build relevant queries.
- It will be assumed that there are a certain number of metadata component sections (i.e., fragments) that are used within multiple object schemas and that would be likely candidates to be reused (and useful for a standardization methodology). For simplicity and to enable cost/benefit analysis, it is assumed these usable metadata stubs will be less than 100.

المنارة للاستشارات
www.manaraa.com

1.4 Approach

The first step in proceeding with this research was to become familiar with the JBI concepts, definitions and structure of the JBI Core Services Reference Implementation (currently Version 1.2). To explore the possibility of creating some type of index or catalog of information objects, it was necessary to understand how they are created and stored. Other previous attempts to model this type of domain were studied.

After learning as much as possible about the problem context and similar areas, a solution was hypothesized that focused on the methods for defining and creating new object types and the storage and retrieval mechanisms of data objects. To implement and test the proposed solution, a sample JBI platform was installed and configured. The database storage system was copied and then modified to conform to the proposal parameters. The storage modifications and other improvements are discussed in Chapter 3. To evaluate the improvements, a test application was developed to compare the proposal to current available methods. This application and preprocessing requirements are detailed in Chapter 4. Evaluation of the method and implementation suggestions are in Chapter 5.

1.5 Research Sponsor

This research is a cooperative effort with its sponsor: AFRL Information Directorate, JBI Branch (AFRL/IFSE) whose mission statement includes "Conducts and sponsors advanced research which directly impacts future joint C2ISR operations by striving to achieve the Joint Battlespace Infosphere vision developed by the AF Scientific Advisory Board" [4].

1.6 Summary

The objective of this research is to improve the quality of the services provided by the JBI. An additional goal was to implement a solution that did not introduce a

www.manaraa.com

great processing burden on the system that handles the storage and retrieval process of published objects, but the focus was on reducing the burden on the users to have a lot of foreknowledge of what they are looking for. This is where the current system falls short—the current user must know what they are looking for and where they can find it. The greater the volume of data that is stored in this framework for information exchange, the more it needs to assist the users in locating the right data to fulfill their information needs.

The remainder of this research is as follows: Chapter 2 contains more thorough explanation of the JBI architecture and other background information. Chapter 3 discusses a performance-enhancing methodology and the approach to the problem solution. Chapter 4 provides the results of the solution implementation. Chapter 5 is an analysis and interpretation of the results. Finally, Chapter 6 provides a conclusion and recommendations for future work.

المنارة للاستشارات
www.manaraa.com

II. Background

2.1 Introduction

In this chapter, background material is presented that pertains to this topic of research. Definitions and emphasis are provided as necessary to provide understanding of some of the key issues.

2.2 JBI

The Joint Battlespace Infosphere (JBI) concept is introduced in Chapter 1. Much of the material here is provided to better illustrate its architecture and critical components.

2.2.1 Key Concepts. A JBI is built on four key concepts [26]:

1. Information Exchange through “publish and subscribe”.
2. Incorporation of military units assigned to the battlespace via force templates.
3. Transforming data into knowledge via data fusion applications (i.e., fuselets).
4. Distributed collaboration through shared, updateable knowledge objects.

An illustrative view of these concepts is shown in (Figure 2.1).

An explanation of these concepts follows (derived from [26]):

2.2.1.1 Information Exchange through Publish and Subscribe. Users and client programs can publish information to a JBI where it is stored as an object. Objects contain both their representative information (e.g., text, image) and its metadata (describing the object). Subscriptions contain search predicates on metadata fields. When an object is published to a JBI, platform subscribers receive this newly published object. The subscription server may deliver the object to a program or a user. The form of the delivered object will depend on whether the subscriber is a user or a program.

المنارة للاستشارات
www.manaraa.com

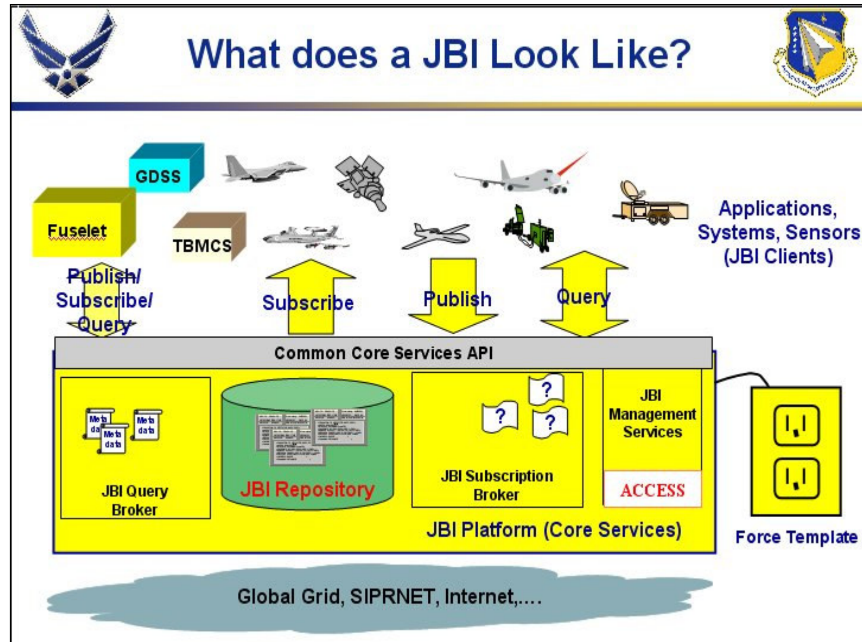


Figure 2.1: JBI Concepts [16]

2.2.1.2 Unit Incorporation via Force Templates. Force templates are the standardized information exchange templates for publish and subscribe for a specific type of unit (i.e., one of the units of "force" in the battlespace). These templates include standard descriptors which are required by every unit and then unit-specific attributes unique to that specific unit (e.g., fighting or support).

2.2.1.3 Transforming Data into Knowledge via Fuselets. A program that subscribes to data in the JBI is called a *fuselet*. A fuselet takes one or more information objects and manipulates them to produce new information. In this respect a fuselet is a scaled down version of an applet or servlet, with the specific task of performing *data fusion*. Data fusion can exploit multiple individual data components by creating a more useful information object through data integration. This is especially useful for increasing the reliability of data through redundancy, processing multi-sensor data for increased spatial or temporal coverage and integrating complementary information to produce information gain [18, 20]. The transformed information may be in the form of one or more new information objects that are sub-

المنارة للاستشارات
www.manaraa.com

sequently published to a JBI. Fuselets fall into different categories (Some are listed below):

1. Fuselets can *transform* data through manipulations such as filtering, sorting, or generalizing.
2. Fuselets can *aggregate* similar data (unify information objects of the same type).
3. Fuselets can *integrate* data (unify multiple different information objects into some kind of single type).
4. Fuselets can *mediate* data (bring into equal membership diverse types and/or formats of information).

2.2.1.4 Distributed Collaboration through Shared, Updateable Knowledge Objects. Users interact with a JBI in many different ways. At a command center, an entire model of the battlespace can be represented on a planning display, while out in the field a soldier may have a personal digital assistant with which he can report the presence of an enemy tank. After the soldier publishes the tank's presence, the command center display may be updated by manipulations of the data received in response to a subscription.

An initiative to support disadvantaged (e.g., systems with limited ability to process complex data) and legacy C^2 nodes in their participation in the JBI net-centric concept has been initiated. This effort, Insertion of Embedded Infosphere Support Technologies (IEIST), is being explored by Air Force Research Laboratory (AFRL) with support from The Boeing Company. IEIST allows the integration of embedded tactical systems into a JBI through the support of a re-locatable off-board software agent (called a Guardian Agent (GA)). A Host Agent (HA) will reside as a "thin client" (i.e., a very small client supported by the bulk processing of a host server) on the tactical platform to act as an interface between the legacy system and the GA. IEIST Force Templates extend the JBI Force Template concept. However, unlike force template definitions, the information needs and generation capabilities

المنارة للاستشارات
www.manaraa.com

of *these* communication systems will include the information the GA needs to assist with the information exchange [24, 25].

2.2.2 Architecture of Reference Implementation. The JBI Reference Implementation (RI) is the Air Force Research Lab's implementation of the JBI vision detailed by the Scientific Advisory Board. The first release to offer object persistence was Version 1.0 (March 2003), which had the following structure:

- Information Object Model Version 1.0
- Common API (CAPI) Version 1.0
- Information Dissemination Infrastructure (Publish/Subscribe/Query)
- MSR and IOR, in both MySQL and Oracle versions
- Security Infrastructure
- MSR and Security Administration

Subsequent releases have redesigned some architectural components and improved capabilities. They have also added sample clients implemented using the CAPI methods. The release used for this research was Version 1.2 (October 2004). Version 1.2 is built on top of the Java 2 Enterprise Edition (J2EE) using the open source JBoss Version 3.2.3. MySQL (4.0.20 or greater) is the underlying default Relational Database Management System (RDBMS) (Oracle 9i/10G is also supported as an option).

2.2.3 Information Objects (IOs). Items of data stored in a JBI Information Object Repository (IOR) are in the form of Information Objects (IOs). Hereafter in this document, an information object will be referred to as an IO or multiple objects as IOs. IOs have a particular structure (e.g., text, image), content (termed its payload) and a metadata representation (that is, data about the data).

There is an important distinguish between an *IO* and an *IO type*. An IO type is the defined format for an IO. Analogous to object-oriented definitions, an IO type

المنارة للاستشارات
www.manaraa.com

is like the object class definition and an IO is an instance of a class. Each IO type has a fixed metadata schema format. These schemas reside in a Metadata Schema Repository (MSR). IOs can extend to other IOs in a parent-child tree structure. These schemas currently have no predefined structure other than a core set of base metadata for all objects (currently Version 1.0 Information Object Metadata Standard) and each IO type has its own fixed schema across all objects of the same specific type. Since there are no defined namespaces other than the base metadata, there is no catalogue or index of metadata tags, and no prescribed correlation between object types. Metadata schemas are defined in an extensible markup language (XML) format. IOs and their metadata are stored in a relational database format wherein the relative paths within the schema are used as row identifiers (hashed to a value based on each unique path) and the value of the tag at that path is stored at that row location in the table. Although this is an awkward format for XML document storage, it is a very common method because relational databases are so prevalent and the stored documents do not need to be parsed to perform searches. Of course, there still exists the effort involved to parse and store the data.

All IO definitions start with the base IO structure. In this sense an implicit hierarchy of IOs is in place. The goals of the object type hierarchy include [9]:

- Organizing the set of types to facilitate human and machine searching of the related metadata elements
- Simplifying the expression of policies over the type hierarchy, and
- Providing a mechanism for obtaining all child subtypes by subscribing to a parent type.

The other structural hierarchy for IOs is built into the MSR, in which IO schemas are stored in a tree-like directory with one or more *packages* containing one or more IO types per package. The segregation of object types by packages allows each Community of Interest (COI) to define shared IO types of interest for their organizational

www.manaraa.com

unit, within a common JBI platform [9]. Essentially, these packages are just a file-like directory structure.

Each IO type is composed of a set of metadata elements in an XML *.xsd schema (See Section 2.3.2 for more on XML Schema). These metadata elements can be any valid XML elements except the base object metadata format which is fixed for every platform. The base IO metadata schema is shown in Figure 2.2. These

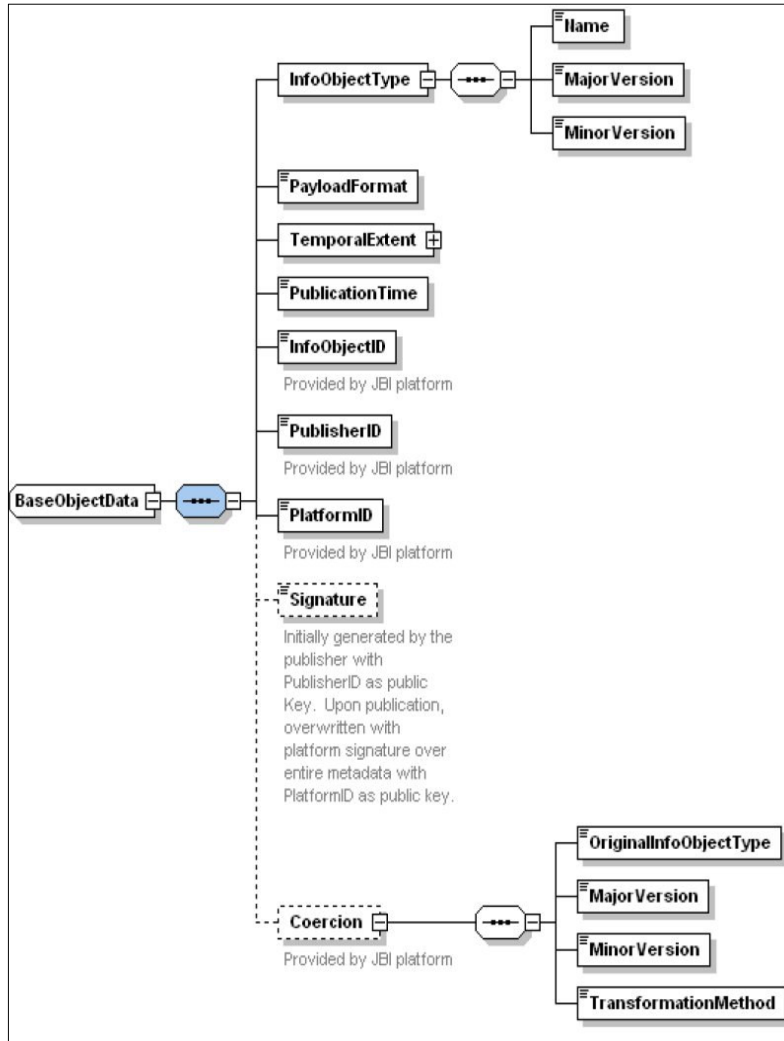


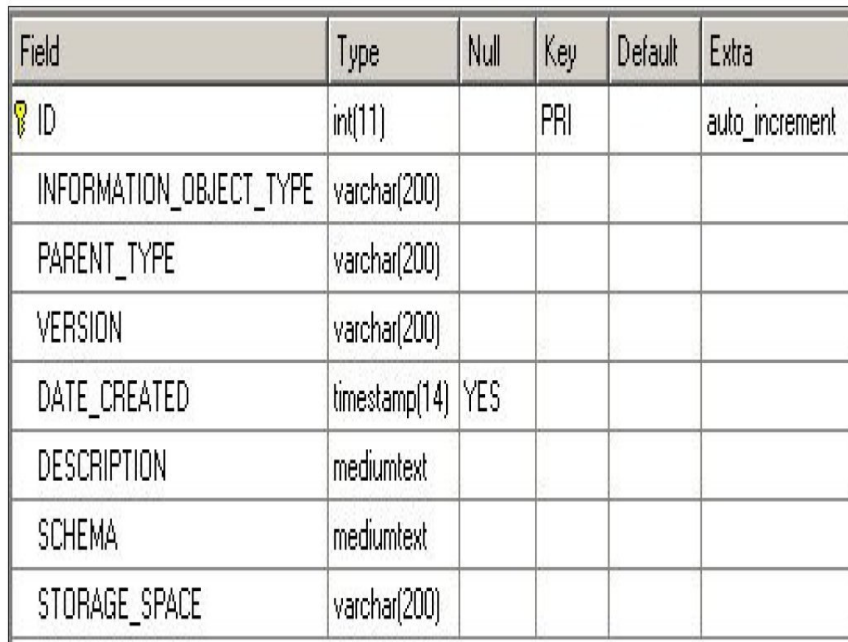
Figure 2.2: Base Information Object Metadata Elements [2]

elements represent the only required metadata in any schema. Any other object type metadata elements are at the sole discretion of a JBI user who has permission to add schemas to a JBI platform. This conformity is enforced by the platform at object

المنارة للاستشارات
www.manaraa.com

type creation time. Thus, the base object elements are the only IO type metadata standardization across all JBI platform implementations.

2.2.3.1 Persistence. The platform maintains a client account and access privileges store as well as a separate MSR and IOR using the underlying RDBMS. IOs have separate storage for their metadata and payloads. The MSR stores a schema using the table shown in Figure 2.3. One IO type is stored per row and the entire XML representation of the schema is stored in the "schema" field. The JBI platform interface for viewing schemas is shown in Figure 2.4. This platform interface con-



Field	Type	Null	Key	Default	Extra
ID	int(11)		PRI		auto_increment
INFORMATION_OBJECT_TYPE	varchar(200)				
PARENT_TYPE	varchar(200)				
VERSION	varchar(200)				
DATE_CREATED	timestamp(14)	YES			
DESCRIPTION	mediumtext				
SCHEMA	mediumtext				
STORAGE_SPACE	varchar(200)				

Figure 2.3: MSR Table

tains links to the full text for each schema in a tree structure by package. When the MSR is populated with many schemas, this interface loads very slow. This is likely due to a combination of factors: the large data type for the schema column in the MSR database table limiting the number of rows that may be returned (larger row sizes means less rows can fit in cache blocks), machine memory limitations, and the potentially large size of individual schemas.

للاستشارات المنارة
www.manaraa.com

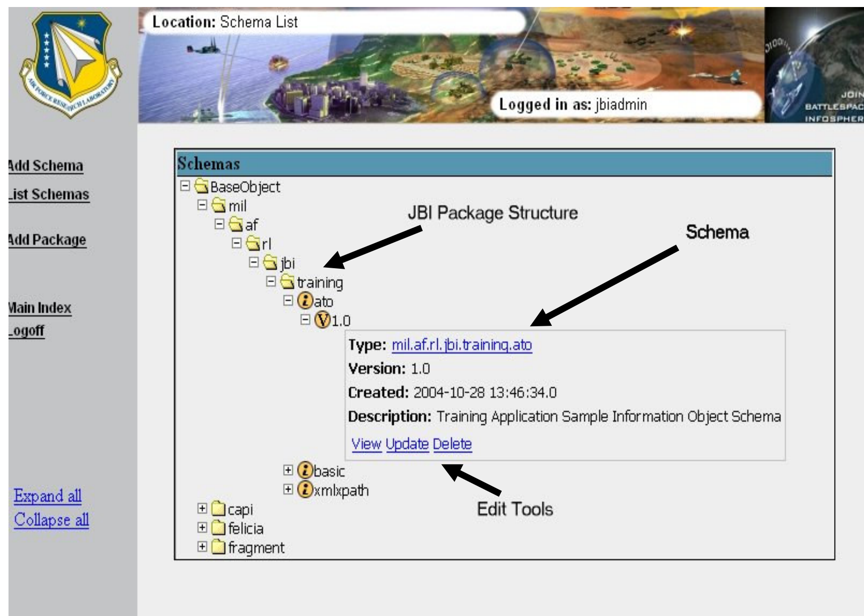


Figure 2.4: MSR Schema Viewer

The IOR storage process and structure is more 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. Hash values are prepended with the string "ior" and negatives are changed to underscores (because databases do not allow negative signs in column names). 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). To illustrate the mapping assignments, a sample schema instance with its corresponding hash map (by node number) is shown in Figure 2.5. A sample IOR table is shown in Figure 2.6.

The 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

المنارة للاستشارات
www.manaraa.com

```
<metadata xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="C:\jbi\afrl\v1.2\schemas
\fragswithfullschemas\fragSchema0.xsd">
      <geospatial_frag>
            <lowLat>50</lowLat>                        <-- 1
            <highLat>100</highLat>                     <-- 2
            <lowLong>20</lowLong>                      <-- 3
            <highLong>40</highLong>                    <-- 4
            <lowElevation>15</lowElevation>            <-- 5
            <highElevation>1000</highElevation>        <-- 6
            <visibility>4</visibility>                 <-- 7
            <humidity>5</humidity>                     <-- 8
      </geospatial_frag>
</metadata>
```

(a) Schema Instance

XML Metadata Path	Node #	Hashed Database Column Name
geospatial_frag\lowLat	1	ior1639392141
geospatial_frag\highLat	2	ior_159236357
geospatial_frag\lowLong	3	ior_718437810
geospatial_frag\highLong	4	ior_641346400
geospatial_frag\lowElevation	5	ior_1214708277
geospatial_frag\highElevation	6	ior1263621817
geospatial_frag\visibility	7	ior340848628
geospatial_frag\humidity	8	ior1592171029

(b) Schema Instance Hash Map

Figure 2.5: Sample Schema Instance With Hash Map

www.manaraa.com

predicate matching in the table. At the onset of this research, other tools were being explored to alleviate this problem. For the purposes of having continuity within the context of this problem study, only the current method was used in this research.

Field	Type	Null	Key	Default	Extra
ID	int(11)		PRI		auto_increment
ARCHIVETIME	datetime	YES			
ior1885786335	int(11)	YES			
ior798735781	int(11)	YES			
ior_267606343	int(11)	YES			
ior1912203471	int(11)	YES			
ior_136733417	int(11)	YES			
ior_1990661973	mediumtext	YES			
ior_164897989	mediumtext	YES			
ior864841027	mediumtext	YES			
ior_1550356851	mediumtext	YES			
ior1598936549	mediumtext	YES			
ior948569701	mediumtext	YES			
ior_2064001258	mediumtext	YES			
INFOOBJ	mediumblob	YES			

Figure 2.6: IOR Table

2.2.3.2 Information Object Versioning. IO type *Versioning* has been envisioned since the introduction of the IO type in the Air Force Scientific Advisory Board JBI reports. This capability will allow a client to request a specific IO type version of which it has knowledge. Version elements have already been included in the base object metadata. An optional element also envisioned and added to the base object structure; *Coercion*, allows a translation (if allowed) from one version to

المنارة للاستشارات
www.manaraa.com

another to allow greater information interoperability. These capabilities are not yet integrated into the JBI RI.

Since this research is focused on the IO type structure and storage mechanisms, methods for versioning and coercion are critical components. As there are no standards or methods in place to address these functions, a proposal for their integration is included in Chapter 3.

2.2.3.3 Information Object Inheritance. Inheritance to support the objectives described in Section 2.2.3 is described in [2] as descending all IO types from the base object type using extension capabilities of the XML Schema specification. An example illustration in a JBI context is shown in Figure 2.7.

```
<xsd:complexType name="Geospatial">   <--------
  <xsd:complexContent>
    <xsd:extension base="BaseObject">   <--------
      <xsd:sequence>
        <xsd:element name="Geospatial" type="GeospatialData"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="GeospatialData">
  <xsd:sequence>
    <xsd:element name="CoordinateSystem" type="xsd:string"/>
    <xsd:element name="Latitude" type="xsd:double"/>
    <xsd:element name="Longitude" type="xsd:double"/>
    <xsd:element name="Altitude" type="xsd:double" minOccurs="0"/>
  </xsd:sequence>
</xsd:complexType>
```

Figure 2.7: Inheritance Using Extension [2]

In this example, "Geospatial" is a new type created by extending from "BaseObject". The additional elements added are the "GeospatialData" elements. This extensibility of XML is perfect for allowing structured data design. However, there are many problems with this approach (which suggests spiraling inheritance from the base object on downward), some of which have been detailed in [21]. These obstacles are

المنارة للاستشارات
www.manaraa.com

discussed further in Chapter 3, where an engineering methodology is proposed for the IO structure.

2.2.4 Common Application Programming Interface. The Common Application Programming Interface (CAPI) (draft version) provides client developers with a single, common interface to any JBI platform implementation. This allows for the development and deployment of several different platform implementations without impact to JBI clients and the information objects they exchange [5].

2.2.5 Information Dissemination. Information exchange and IO persistence are managed by the JBI platform. The platform maintains an MSR and IOR of metadata schemas and IOs (respectively). Before allowing a client to publish, subscribe or query, permission is requested to verify the IO type and version details against the user rights. If the user is granted access, a publisher sequence, subscriber sequence, or query request is executed.

2.2.6 Information Retrieval. A useful Information Retrieval (IR) system should address the efficiency with which it matches users to documents (or in this case, IOs) based on their information needs. The development thus far of the JBI core services has not yet adequately addressed some important aspects of IR. This research will attempt to bridge these gaps.

IR efficiency can be viewed as a process of determining the degree to which an information need is filled by an information request. In a JBI, the mechanisms of information dissemination (publish/subscribe/query) are only part of the process. Schemas have elements that can be matched using relative data paths, but the JBI platform architecture does not lend itself to efficient searching using these predicates. The one table per IO type structure of the IOR requires either:

1. a union of all IOR tables, or
2. an iterative IOR table search

المنارة للاستشارات
www.manaraa.com

to find every IO type with the same relative metadata path. Even if the platform service subscription method did not restrict a subscription to a single IO type, this type of searching is wholly inefficient.

The structure of the metadata representations of IOs are ideally suited to build a framework for relating IOs in some topical or semantic way. An exploration of the metalanguage used to describe IO types is the topic of the next section.

2.3 XML

Extensible Markup Language (XML) is a simplified form of the Standard Generalized Markup Language (SGML) [13]. It is a meta-markup language that was created to describe data. Unlike the Hypertext Markup Language (HTML), which was designed to describe how content should be displayed on a web page (using predefined tags), XML was designed to describe the data content itself.

2.3.1 XML Rules. XML rules of syntax are very strict and thus, very simple to use. For this reason, parsing software has been very easy to create and use. XML documents that adhere to these syntax rules are said to be *well-formed* (see Figure 2.8). The basic rules are:

- The first line of an XML document describes the XML version and encoding.
- Tagged document items are called *elements* and elements can have *attributes*.
- The next line after the version and encoding must be the root element of the document, and all elements must have a closing tag.
- All other elements must come between the root element opening and closing tags and must be properly nested.
- XML element names are:
  - case sensitive,
  - can consist of letters, numbers, and other characters,

www.manaraa.com

– cannot start with a number or punctuation character or the letters xml (in any case), and

– cannot contain spaces

- Attributes of elements must always be quoted.
- White space is preserved (unlike in HTML.)
- Comments are written as shown.
- Elements can have mixed or empty content.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- This is a comment -->
- <root>
  - <child attributeName="inQuotes">
      <subchild>.....</subchild>
    </child>
  </root>
```

Figure 2.8: XML Basic Syntax

2.3.2 XML Schemas and DTDs. XML documents can be made to conform to a specific structure through the use of a defined XML Schema format or Document Type Definition (DTD) format. Each of these models can be used to validate an instance XML document against a prescribed format, but the advantages of using XML Schemas over DTDs make the choice of using XML Schema an easy one. Some of these advantages are:

- XML Schema allows the use of more than 44 data types versus only 10 for DTDs.
- XML Schema allows more constraints to be placed on data types to more strictly define a specific format (e.g. users can be forced to represent a decimal number

المنارة للاستشارات
www.manaraa.com

to 2 fractional places for a currency element to pass validation constraints in an instance document).

- The syntax is the same as for XML instance documents and thus, much easier to read and understand.
- Types can be extended or restricted as in an object-oriented sense (critical to applying inheritance to the IO structure).
- XML Schema allows elements to have null content (may be critical to allow JBI cross platform conformity to standardize certain base schemas).
- XML Schema allows multiple same named elements, but with different content.
- XML Schema allows substitutable element names (this will allow a COI to redefine an acceptable standardization to conform to their vernacular)

2.3.3 XML Namespaces. The World Wide Web Consortium (W3C) envisioned reuse and modularity of XML schemas in software modules. For this reason, the mechanism of *XML namespaces* was created to qualify attribute and element names with a Uniform Resource Identifier (URI) [10]. Namespaces are useful for avoiding collisions when there might be multiple commonly named elements in different namespaces because these elements can be qualified with a namespace prefix and colon before the element name (e.g., <my-namespace:element-name>). Furthermore, schema file names which contain appended version numbers can be used to distinguish between differing schema versions of the same IO type (e.g., myschema_1_0.xsd and myschema_2_0.xsd to represent versions 1.0 and 2.0 of myschema).

2.3.3.1 Namespace Methods. The W3C specification only provides the definition of and guidelines for declaring namespaces. Many of the W3C contributors, as part of the xml-dev list group, have been developing a set of XML Schema best practices to assist developers with choosing the best way to use namespaces in their projects that will deal with multiple schemas. This type of guidance is rightly not addressed in the namespace definition documents, but is needed for projects which will

المنارة للاستشارات
www.manaraa.com

Table 2.1: Namespacing Methods

Method	Description
Heterogeneous	Give each schema a different targetNamespace
Homogeneous	Give all schemas the same targetNamespace
Chameleon	Give the "main" schema a targetNamespace but no targetNamespace to "supporting" schemas (supporting schemas will take on targetNamespace of main schema, like a chameleon)

contain many schemas (like this research). One result of this collaborative effort has resulted in the definition of three different namespacing approaches. A combination of these approaches may be suitable for this research, although the issue of namespaces has not yet been addressed in the JBI Reference Implementation (RI). Table 2.1 and the example illustrations that follow are adapted from [8].

The project undertaken will dictate whether one of these methods or a combination of these methods will be used. Since the latest version of the JBI RI does not support custom namespaces, this research includes suggestions for the incorporation of namespace methods into a later release. To aid in the understanding OF the various namespacing techniques, Figure 2.9 provides a simplified illustration of the three methods.

Heterogeneous

- Use when there are multiple elements with the same name. (to avoid name collision)
- Use when there is a need to visually identify in instance documents the origin/lineage of each element/attribute. In this design the components come from different namespaces, so you have the ability to identify in instance documents that "element A comes from schema X".
- Note in the heterogeneous example that namespaces "Q" and "R" both define "Proxy" schemas which are used in the "Z" namespace schema. The multiple

www.manaraa.com

namespaces allow both formats of this similar component to be declared and used inside the same schema.

Homogeneous

- Use when all of the schemas are conceptually related
- Use when there is no need to visually identify in instance documents the origin/-lineage of each element/attribute. In this design all components come from the same namespace, so the ability to identify in instance documents that "element A comes from schema X" is lost. In those situations where it is not required to categorize elements/attributes differently, this design approach is well suited.
- In the homogeneous example, both component schemas and the integrated schema with the "include" declarations are from the same "Library" namespace.

Chameleon

- Use with schemas which contain components that have no inherent semantics by themselves,
- Use with schemas which contain components that have semantics only in the context of an including schema,
- Use when it is not necessary to hardcode a namespace to a schema, rather the goal is for including schemas to be able to provide their own application-specific namespace to the schema.
- Note that there are no namespace declarations in the chameleon example "Q.xsd" and "R.xsd" schemas, so any type definitions in these schemas are in a sense "coerced" to the "Z" namespace.

As a rule of thumb, if a schema just contains type definitions (no element declarations) then that schema is probably a good candidate for being a Chameleon schema.

www.manaraa.com

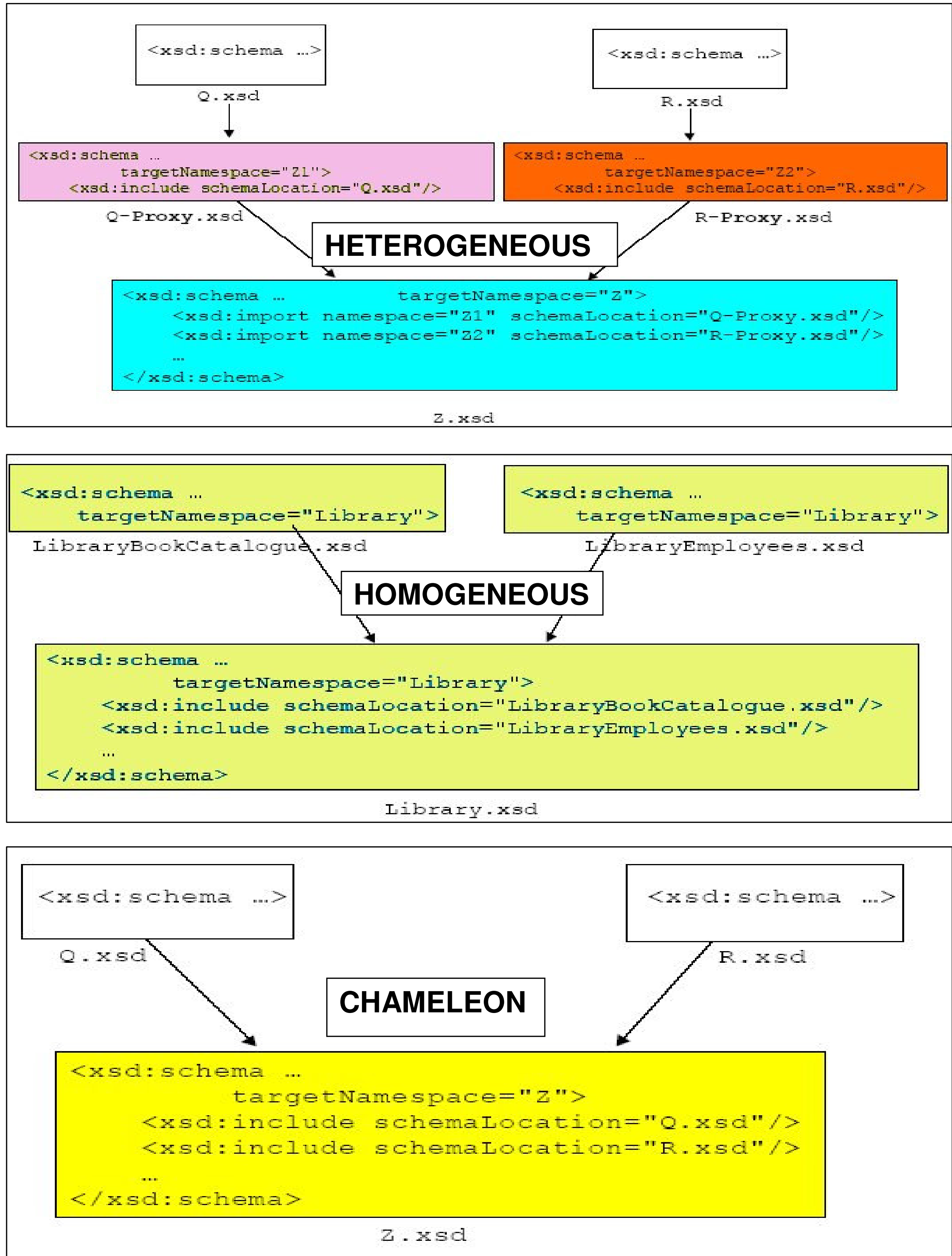


Figure 2.9: Namespacing Methods [8]

المنارة للاستشارات
www.manaraa.com

When using the chameleon approach with element declarations, there is always a possibility of name collisions if the data type of any complex type within the integrating schema is defined in more than one of the "included" or "imported" schemas. In the provided chameleon example, assume Q.xsd and R.xsd each contain a unique complex type definition named "PersonType". If an element is then declared in the Z.xsd schema which is of this data type, there will be a collision on this element.

If the JBI RI is to take advantage of namespaces, it should use heterogeneous methods, as this allows the greatest flexibility for multiple platforms. Not only will this allow clients to declare identically named data types within their own platforms, but they will be able to use the commonly defined data types of a central JBI namespace. The benefit of central namespace definitions is that IOs from multiple platforms will conform to a single type definition and there can be greater cross platform data sharing and integration.

2.3.4 XPATH. XPath is a non-XML language used to identify particular parts of XML documents [15]. An XML document is represented as a node tree and the XPath language allows the selection of any node by element traversal. Seven kinds of nodes are recognized [15]:

- The root node
- Element nodes
- Text nodes
- Attribute nodes
- Comment nodes
- Processing instruction nodes
- Namespace nodes

The basic syntax of XPath resembles file system addressing. The node-searching expression in the XPath language is the *location path*. An absolute path (from the root

www.manaraa.com

node) is represented as "/" and an all-descendants selection (relative and absolute) is represented by "//". As in file system addressing, a ".." represents the parent node and a "." represents the current node. Several wildcards are also available to build these expressions.

XPath expressions are used to build predicates by using relational operators to match element and attribute values at the location specified by the XPath expression. The *expression* finds the matching node paths and the *predicate* tests the node (as a filter) for a match. The process continues until the predicate is tested against all matching nodes.

2.3.5 XML Inclusion Methods. There are two useful XML techniques for building larger schema documents from smaller modular components. These methods are considered in this research for their usefulness as it pertains to building IO types from smaller schema segments.

The XML Inclusions (XInclude) Version 1.0 W3C Candidate Recommendation defines a namespace associated with the URI http://www.w3.org/2001/XInclude (17 September 2002) [6]. The XInclude namespace contains two elements with the local names *include* and *fallback* [22]. The syntax allows multiple inclusion references to other XML documents. If the referenced inclusion element is not available, the fallback element can specify default behavior. Essentially, using an XInclude reference within an XML document is analogous to using a "#include" in the C programming language. When the included item is fetched and processed, the section of the parent document with the include is just replaced with the fetched document (or section of document using another mechanism, XPointer). This recommendation is still under development and there are limited parser tools available to handle the overhead of processing the includes. Furthermore, many of the papers, example documents and tutorials available on XInclude use XML document development and not schema development. This may be due to the fact that XML Schema has its own mechanism

المنارة للاستشارات
www.manaraa.com

for inclusion. For these reasons, the XML Schema include and import methods are preferable for this research problem.

The XML Schema Part 0: Primer Second Edition W3C Recommendation (28 October 2004) has two elements specifically available for including schemas within other schemas [7]. The *include* element allows the reference to another schema within the same target namespace as the main schema. The *import* element allows inclusions of schemas from other namespaces. As with XInclude, it is the responsibility of the schema processor to insert the included and/or imported schemas before attempting instance validation. But unlike the XInclude method, there are many processors available that recognize the XML Schema syntax of include and import (such as Xerces) and will perform the preprocessing as part of the validation.

2.3.6 XML Validators. A *valid* XML instance document is one that conforms to the target schema declared (if any) in that document. An example of this declaration is *xsi:schemaLocation="http://www.myjbinamespace.mil"*. If schema validation is enforced in an application, instance documents (*.xml) are compared to the referenced schema document (*.xsd). When a document does not conform, the application can reject the instance outright or correct it. Validation checks are made to both form (the proper number and ordering of elements and attributes) and content (valid element data type values). In addition to Xerces, there are many other free XML Schema parsers and validators.

2.3.7 Distributed Schema Design. XML Schema has several methods for mimicking inheritance through extensible schema design. Leveraging these tools allows for distributed schema design, unlimited element and attribute vocabularies, and aggregation of data for semantically related topics. Distributed schema design allows multiple users (or teams of users) to work independently to develop and control the evolution of smaller modular schema components or user-defined data types. Users who find a need for a small schema component with or without some minor modifica-

المنارة للاستشارات
www.manaraa.com

tions can reuse these pre-defined data types (or derived data types using *restriction* or *extension*) on included or imported schema elements.

Using a restriction base, a new type can be declared that either eliminates some elements from the base type, or restricts the range of values or number of instances of an element allowed in an instance document. Extension allows the use of the base type with additional elements added to the new type. Extension most closely resembles object inheritance in the classic sense. Figure 2.10 provides a simple illustration of restriction. In this example a "Publication" type was initially designed to allow multiple authors. A restrictive type was extended from this to "SingleAuthorPublication" which removed the unbounded value of the maxOccurs attribute from "Author" (the unbounded property allows multiple instances of an element in an instance document, the default is one instance). The "ZeroAuthorPublication" restriction also extends from the "Publication" type and eliminates the "Author" element completely. Extension was first illustrated in Figure 2.7.

```
<xsd:complexType name="Publication">  <------
    <xsd:sequence>
        <xsd:element name="Title" type="xsd:string" maxOccurs="unbounded"/>
        <xsd:element name="Author" type="xsd:string" maxOccurs="unbounded"/>
        <xsd:element name="Date" type="xsd:gYear"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name= "SingleAuthorPublication">  <------
  <xsd:complexContent>
    <xsd:restriction base="Publication">  <------
      <xsd:sequence>
        <xsd:element name="Title" type="xsd:string" maxOccurs="unbounded"/>
        <xsd:element name="Author" type="xsd:string"/>
        <xsd:element name="Date" type="xsd:gYear"/>
      </xsd:sequence>
    </xsd:restriction>
  </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name= "ZeroAuthorPublication">  <------
  <xsd:complexContent>
    <xsd:restriction base="Publication">  <------
      <xsd:sequence>
        <xsd:element name="Title" type="xsd:string" maxOccurs="unbounded"/>
        <xsd:element name="Date" type="xsd:gYear"/>
      </xsd:sequence>
    </xsd:restriction>
  </xsd:complexContent>
</xsd:complexType>
```

Figure 2.10: Restriction [12]

Another tool that allows great flexibility in schema development is the use of the **xsd:any** element. Adding this element at the end of a schema allows an in-

المنارة للاستشارات
www.manaraa.com

stance document to include any unnamed element to an instance document and still pass validation. There is also a similar mechanism for allowing additional attributes. Figure 2.11 shows this element added to a book schema. Setting minOccurs = "0" designates that adding an element is optional.

```
<xsd:element name="Book">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="Title" type="xsd:string"/>
            <xsd:element name="Author" type="xsd:string"/>
            <xsd:element name="Date" type="xsd:string"/>
            <xsd:element name="ISBN" type="xsd:string"/>
            <xsd:element name="Publisher" type="xsd:string"/>
            <xsd:any minOccurs="0"/>  <------
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

Figure 2.11: Using xsd:any for Optional Schema Elements [12]

2.4 DOD Metadata Registry

The Department of Defense (DOD)has established a registry [3] to promote standardization of metadata across all branches of the military. The registry has been made easily accessible to users at all levels to facilitate shared data exchanges by conformity to the XML formats. There is representation from many areas such as contracting, logistics, aircraft, etc.

Although is a useful tool, perusal of a small sample of schemas in the registry immediately showed that each community was using a different representation standard. For example, one schema used a combination of Resource Description Framework (RDF) and Web Ontology Language (OWL), which are standards for describing resources on the web. Other schemas were in DTD format and still more were using

www.manaraa.com

XML Schema. Since each format must be processed differently, it is not so simple to integrate such a disparate collection of schemas into a platform that has been designed to process data in a single format (in this case, XML Schema). Because the registry proposes no single format for its nearly 60,000 elements and greater than 2,300 schemas, the information will not be easily exported to other domains. There is a downloadable Microsoft Access database that contains many (but not all) of the elements. An application can be designed to export these elements into a standard format (such as XML schema) where they would then be more useful to the JBI community, as a starting point for defining a basis library of schemas.

The DODMDR is not the only place where common metadata definitions have been explored. There is a working group with members of more than a dozen Air Battle Planning organizations, who have a working draft of a Common Mission Definition (CMD) Information Model for their community and namespace [17]. In their latest draft, they have defined 88 XML metadata complex types related to Air Battle Planning. This information model is also a useful place from which to extract a common metadata library.

2.5 Summary

This chapter contains an examination of applicable background information that was explored to achieve the goals of this research. Specifically, it covers the JBI components, architecture and information exchange mechanisms. Then there is an exploration of the XML metalanguage and some of its components, standards and restrictions. Using what can be learned from this relevant information, specific methods are used in this research proposal or are included in the implementation recommendations.

The remainder of this document contains the methodology proposal (Chapter 3) and the implementation (Chapter 4) of the recommendations used to alleviate the problems addressed by this research endeavor. Chapter 5 provides an analysis

المنارة للاستشارات
www.manaraa.com

of the tests used to evaluate this proposal. Finally, the conclusion and future work recommendations are contained in Chapter 6.

المنارة للاستشارات
www.manaraa.com

III. Methodology

3.1 Introduction

The goal of this research is to improve the quality of service delivered by a JBI by introducing a new information engineering framework for Information Object (IO) schemas and thereby improve IO retrieval within a JBI. In Chapter 1, the IO is introduced as the basic unit of data within the JBI. There are limitations associated with the basic construct of an IO schema, as it is currently defined. The impact of these limitations imposes a great burden on clients (who create schemas for the objects, publish objects, subscribe to objects and/or query for objects) and on the JBI core services (which handle the overhead of storing the published objects and returning the stored objects to clients based on subscription and/or query parameters). This chapter expands on the ramifications of these limitations as it pertains to the quality of information retrieval. It then proposes a solution that will alleviate the burden imposed by the current IO schema structure. After the problem is defined and the solution approach is outlined, there is a discussion of evaluation techniques and predicted outcomes.

3.2 Problem Definition

The primary problem addressed is that there is no common methodology, framework or standardization for structuring and defining new IO types. In addition to defining an approach to remedy this shortfall, several other significant improvements are realized. These are:

1. A technique for searching across related objects,
2. More effective use of database storage space in the Metadata Schema Repository (MSR),
3. Reduction in the time needed to build schemas, and subscribe to and/or query for objects,
4. Elimination of overhead associated with introducing revisions to object schemas,

المنارة للاستشارات
www.manaraa.com

5. Definition of standards and methods for versioning and coercion,
6. Reduced IO type knowledge required by users to do a more thorough search of available objects, and
7. Promotion of the use of namespaces to allow standardization and reuse of base schema components across multiple platforms.

3.3 Primary Objective: Common IO Engineering Framework

IO structures are defined and registered in the MSR. The actual IO comprises both the metadata and the payload. Subscriptions and queries are matched against the metadata fields of the IOs. Section 2.2.3 discusses the goals of the IO hierarchical structure. The implementation thus far has not taken advantage of the extensibility capabilities of XML schema design. The primary benefit of the package structure as implemented (i.e. IO type schemas grouped by packages) is that it is used for imposing policy for access rights, but it is not ideally suited to perform the additional duty of relating objects semantically.

A significant problem with the existing hierarchical structure is that it is difficult to construct a hierarchy of IO types using single inheritance. XML schemas are not structured to support inheritance. A complex type *within* a schema can be extended (which is similar to object oriented inheritance), but this does nothing to support inheriting from full schemas which may be composed of multiple complex types. There also may be no clearly defined parent in a schema composed of multiple types, so it is difficult to say which type should be the parent.

For example, assume there are two complex types within an IO type schema: one containing geospatial data and the other containing intelligence report data. The hierarchical structure states that one will be defined first and extend from the base object (as defined in Section 2.2.3.3). Then the next type will extend from this new type, etc. Which of these components should be the parent? Will this be easier to define for a given context? If the context is multiple reports by a stationary

المنارة للاستشارات
www.manaraa.com

intelligence unit, the *report* data may be the parent type (because the geospatial unit is unchanging and may be just an attribute in this context), but if the report contains satellite imagery from a U-2 spy plane, perhaps the *geospatial* data is more critical and should be the parent. In either case, individual communities of interest (COIs) may define this in either way and it immediately becomes impossible to retrieve this possibly related data with a single query or subscription. Furthermore, neither community may be aware of the other's IO type definitions.

Multiple inheritance is also a problem. How is any JBI platform to relate common IO types which may inherit from more than one IO type? The package structure of IO type schemas is insufficient because there may be an IO type that requires some form of multiple inheritance from different components. Which package should it reside in and who will make this decision? Even if a COI has an appointed information manager who makes and enforces standards within that platform, how will other platforms be able to find and subscribe to similar IO types on this platform?

These questions illustrate that it is more likely that these geospatial and report *components* should be "included" elements in an IO type, instead of either one being a parent to the other.

The solution to the problems implied by these questions first involved redefining the structure for IO type schema definitions to allow for component based schema development. Thus, a framework is introduced by which IO type schemas are constructed from XML schema components.

For this work, the basic unit of information is called a *fragment*, and object schemas are composed of multiple fragments. For example, all object schemas are to consist of a base object fragment (the first defined fragment). Other fragments with a likelihood of reuse (e.g., geospatial data) will be defined.

المنارة للاستشارات
www.manaraa.com

3.4 Improvements

Evaluating the improvement of the introduction of a common OI engineering methodology can best be achieved by measuring the improvement in the other areas impacted in the supporting objectives. Specific improvement areas are elaborated below.

3.4.1 Multi-Object Search. Even though an obvious advantage of the IO metadata structure would be for queries and subscriptions to search across multiple object types (with some related metadata elements), this capability is not built into the platform or defined in the Common Application Programming Interface (CAPI). Therefore, any client wishing to subscribe to or query for objects with some correlation between them would need foreknowledge of each object's existence and structure. Two different object types may even have tags with the exact same meaning in an informational sense (e.g., tags named "latitude" versus "lat" and "longitude" versus "long"), but because of naming conventions or metadata tag nesting order, they would be totally separate with respect to a search or query. Of course, even if the metadata were identically named and ordered, the capability to search across multiple objects is not built into the architecture of the current version of the platform. The user is required to build one query or subscription for each object type. This, in effect, requires a "smart" client, which is to say, a client who knows exactly what type of information is available for matching his information needs, as well as its structure.

The metadata schema is an opportunity to address the issue of relating IO types, beyond the package structure, by correlating object types by subsets of the metadata they may have in common. Even if IO types are not descendants of the same parent type in the traditional hierarchical sense, they may express or represent semantically related concepts. This relationship should be exploited on to the maximum extent possible while still allowing platforms the least restrictive requirements for IO schema formats.

المنارة للاستشارات
www.manaraa.com

3.4.2 Better Schema Storage Method. The Metadata Schema Repository (MSR) stores the IO schema for each IO type within a JBI platform. The MSR table contains a column to store the full text representation of each schema. If component development is introduced that takes advantage of a way in which IO type schemas may reference similar metadata stub segments, references to these similar components (instead of repeatedly introducing the same segments into different object schemas in the MSR) will require less space within the MSR. The present method of storing the full schema will still be employed, but the schema will be a compact reference of fragment file includes and the element names assigned to their corresponding types. These schemas in the MSR will consist of one or more included components that will be defined and searchable in another table. See Figure 3.2 for an example of this type of schema. The corresponding non-fragment schema (with full fragment definitions), contains 108 lines compared to the new schema which only has 16. This may also result in a time savings when the platform fetches many IO type schemas from the database because the result set size will be smaller and require less memory.

3.4.3 Less Effort to Build, Subscribe, Query. Each time a JBI is stood up, it could potentially require excessive time and effort to define, build and populate an MSR with IO type schemas. Component schema development (using fragments) facilitates many of the MSR population tasks. Object data management development often requires the ability to define objects which contain references to other objects [11]. Similarly, there are likely common elements that need to be included in different IO types. After components are defined and used to compose these multiple schemas, a subsequent change to a component (by the addition, deletion or data-type change of its metadata elements) does not need to be propagated through all the IO types in which this component resides. Without component development, there would have to be some mechanism of cataloging these similar stubs and cascading changes to all schemas which specifically include them, as opposed to including only a reference. Component development will allow a library of common stubs to be re-used by

المنارة للاستشارات
www.manaraa.com

multiple users within a platform (for locally defined components) or across platforms (for generic components from a central datastore).

Subscriptions and query mechanisms also benefit. The current architecture requires a subscriber to know the name and structure of each IO type he wishes to subscribe to. The platform also requires a separate subscription for each of these types. The proposed component method introduces a method of relating similar IO types by their common inclusion of an identical component and a method to include multiple IO types in a subscription or query by selecting a component for the search (rather than the IO type).

The bottom line is that this improvement provides a way of semantically relating IO types that have common components. Although these relations are not a hierarchical taxonomy, it is a useful framework for object type categorization. Relating objects semantically in this manner can allow applications to discover the relationships between objects from seemingly dissimilar packages. For example, suppose that there is a schema for representing a particular battlefield target (including location) and a schema for describing the allocation of friendly troops (including current location). Published objects of these types may only have geographical data in common, but a search for all components with geographical data within a rectangular coordinate system could return both of these objects (without having knowledge of their schema types). This demonstrates the ability to obtain all IO types containing commonly defined geographical metadata components (i.e., latitude and longitude). The application described in Chapter 4 demonstrates this retrieval method by fragment specification.

3.4.4 Simpler Object Schema Revision Rules. Another problem was highlighted with the release of JBI Reference Implementation (RI) V1.2. With this release there was a modification to the base object definition from V1.1. JBI platform configurations have an option for enforcing schema validation. If objects created from the old version are validated with respect to this new base object schema, the vali-

المنارة للاستشارات
www.manaraa.com

dation will fail. Some method of propagation is needed to update changed metadata schemas to allow validation. If the base object schema was merely referenced in all of the object schemas which inherit from it, this would no longer be a problem, since only the fragment schemas referenced would be updated.

The current JBI architecture does not include standards for versioning IO types, including what constitutes a major schema version change versus a minor change. At the time of this research endeavor, the XML community has no standards for versioning because of the complexity of schemas with respect to the data exchange between producers and consumers of XML content. The focus of many suggested rule sets is on whether a change will allow compatibility for existing producers and consumers of data. For a JBI subscribers, a compatible version change would be one in which current subscribers could continue to receive and process newer versions of object types with no change to their applications. For JBI publishers, compatibility would allow their published objects to pass validation even if their content is of an older type than the latest version.

Object oriented database developers address this issue by grouping object versions into a particular *configuration*. A configuration is the version of the entire database at a particular point in time. This is sometimes done automatically by the Object Database Management System [11]. The JBI architecture must consider that users will still have applications configured to deal with older schema versions after IO types have been updated.

3.4.5 Versioning and Coercion Methods. Standards and techniques for versioning and coercion have not yet been implemented or suggested in the latest JBI Reference Implementation (RI). Newer versions of IO types are necessary when there are element or attribute changes, additions or deletions to an IO type. Adding fragments to the mix introduces a complexity that also needs to be addressed. Coercion rules are needed to facilitate the translation of IOs to earlier or later versions. Coercion is a defined translation, per schema element, to convert objects from one

المنارة للاستشارات
www.manaraa.com

version to another. With an emphasis on application compatibility, this research proposal includes definitions for major or minor version changes and suggests coercion techniques. This will allow IO type schemas to evolve without negatively impacting existing subscribers. Since fragment schemas will be the building blocks of IO type schemas, coercion and versioning will be driven by the changes to fragments, and IO type changes will be regulated by these fragment changes. Since the current method of IO storage is relational databases, mechanisms are proposed that utilize the tools available in that access method. Since fragments introduce some IO type standardization, these rules will be much easier to define. These rules will be discussed in Section 3.5.8

3.4.6 Less IO Type Knowledge Required by Clients. In Chapter 1, the notion of how information overload has placed a new burden on the military decision-maker is discussed. A JBI should be able to deliver all the needed information with only the minimum schema knowledge required by the user. Using the geographical location example from Section 3.4.3, if a decision maker wanted to know as much as possible about a particular location grid, and there was a known geographical component used in several schemas with location data, he would simply build a subscription or query using only that component. In the current system, he would need to know the existence and format of every single IO type which contains geographical data. Furthermore, he would need to know how each IO type presented its geographical data, because there is no standardization across platforms (although there may be some standardization within a community of interest, which is unknown to other platforms).

If this example was extended to account for the possibility of cross platform compatibility, the advantages would be even more pronounced. Standard components could be deployed with the JBI RI, and platform developers could be encouraged to use the standardized components. If platform discovery is implemented, users could

المنارة للاستشارات
www.manaraa.com

easily search for information across platforms without developing platform-specific search predicates for each platform.

3.4.7 Take Advantage of XML Namespaces. A particular JBI community of interest (COI) may require that their users follow naming conventions defined within their domain. However, this may not facilitate searching other platforms for similar data (when that capability is eventually introduced). To allow maximum flexibility for information structure implementation, IO type development has been thus far loosely defined and limited to the base object structure.

It may be difficult to enforce conformity to a specific naming or formatting convention for IO type schemas. However, as discussed in Section 2.4, there is a working group drafting the Common Mission Definition (CMD) Information Model for the Air Battle Planning arena and there are many communities participating in the DOD Metadata Registry service. As these efforts illustrate, there is motivation for some standardization. Therefore, it would be beneficial to these communities to prescribe some of their basic metadata structures in common namespaces. These structures could then be reused by the JBI communities that have common information needs. The CMD effort signals a desire to maintain the greatest flexibility for sharing their critical information with other platforms. The CMD working draft already contains many useful XML complex types that would make suitable fragments. Since there are a large number of participating organizations, these fragments could be stored in a central JBI namespace which would allow ease of conformity.

3.5 Solution Approach

The approach to resolving the IO structure shortcoming was to introduce the new IO engineering framework previously described as the primary objective. Then the new methodology was integrated into a sample JBI. In the following sections, this process is outlined and illustrated.

www.manaraa.com

3.5.1 Introduce A Component-Based Schema Structure. In Section 3.3 the fragment schema concept was introduced. The current structure of the MSR can store fragment schemas but the fragments have to be placed in separate database tables to allow separation of fragment types from IO types. Fragments are simply small schemas composed of a single complex type definition (such as a "geospatial fragment). IO type schemas are then composed of one or more frgaments. Several sample fragment schemas have been added to the fragment table, and new sample IO type schemas have been added to the MSR that are composed of these fragments. The new fragment tables have been manually populated with fragment names, fragment schemas and IO type pairs based on the sample object types published (with the assumption that this function will be introduced into the platform upon implementation of this technique). These tables are explained in more detail in Section 3.5.6. The sample fragment schemas are simple and compact and are used for illustrative and testing purposes only. They are not put forth to represent any real world recommendations. The sample fragment schema files geospatial_frag.xsd and target_frag.xsd are shown in Figure 3.1.

The full list of sample fragments used is covered in Chapter 4. Note that these files have XML complex type definitions which are used to build sample schemas. Thus, the fragment files are essentially a library of types. Schemas built from these fragments include named elements which are declared to be elements of these types.

3.5.2 Fragment Naming Conventions. Some standard fragment naming rules have been used. Let **"sample"** be a fragment that is to be created. XML namespace rules require that schemas be stored in a file with a unique name within a namespace to support validation. Therefore, to eliminate any confusion, the major and minor fragment version will be incorporated into the fragment file name. Thus, the naming conventions are:

المنارة للاستشارات
www.manaraa.com

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
	<xsd:complexType name="geospatial_fragType">
		<xsd:annotation>
			<xsd:documentation>geospatial fragment</xsd:documentation>
		</xsd:annotation>
		<xsd:sequence>
			<xsd:element name="MajorVersion" type="xsd:integer"/>
			<xsd:element name="MinorVersion" type="xsd:integer"/>
			<xsd:element name="lowLat" type="xsd:decimal"/>
			<xsd:element name="highLat" type="xsd:decimal"/>
			<xsd:element name="lowLong" type="xsd:decimal"/>
			<xsd:element name="highLong" type="xsd:decimal"/>
			<xsd:element name="lowElevation" type="xsd:integer"/>
			<xsd:element name="highElevation" type="xsd:integer"/>
			<xsd:element name="visibility" type="xsd:decimal" minOccurs="0"/>
			<xsd:element name="humidity" type="xsd:decimal" minOccurs="0"/>
		</xsd:sequence>
	</xsd:complexType>
</xsd:schema>
```

(a) geospatial_fragType complexType

```xml
	<xsd:complexType name="target_fragType">
		<xsd:annotation>
			<xsd:documentation>target fragment</xsd:documentation>
		</xsd:annotation>
		<xsd:sequence>
			<xsd:element name="MajorVersion" type="xsd:integer"/>
			<xsd:element name="MinorVersion" type="xsd:integer"/>
			<xsd:element name="ID" type="xsd:positiveInteger"/>
			<xsd:element name="type" type="xsd:positiveInteger"/>
			<xsd:element name="surfaceArea" type="xsd:positiveInteger"/>
			<xsd:element name="priority" type="xsd:positiveInteger"/>
			<xsd:element name="vulnerability" type="xsd:positiveInteger"/>
			<xsd:element name="threat" type="xsd:positiveInteger"/>
			<xsd:element name="bda" type="xsd:positiveInteger"/>
			<xsd:element name="restrictions" type="xsd:positiveInteger"/>
		</xsd:sequence>
	</xsd:complexType>
</xsd:schema>
```

(b) target_fragType complexType

Figure 3.1: Fragment Schema Files

المنارة للاستشارات
www.manaraa.com

- Fragment versions: As in IO type schema versioning, major and minor version changes should be tracked for fragments (e.g., Version 1.0 denotes Major version 1, Minor version 0).
- Fragment file name: ***sample_frag_majorVersion_minorVersion.xsd***. Example: geospatial_frag_1_0.xsd
- Complex type name defined in fragment file: ***sample_fragType***
- IO type and IO type schema file name: IO type schema names require that the major and minor version be appended to the file name.

IO type and IO type schema file names should be meaningful to a platform community but will not be used for fragment searching. As with the current method of searching by IO type name, this proposal provides methods for searching by fragment names without knowing IO types.

These naming conventions make fragments very flexible because individual COIs can control naming their fragment and IO type schema elements while still following a standard that allows other COIs to subscribe to and query for the known fragment structures. Since thoughtful consideration has already been given to the battlespace elements in the DOD Metadata Registry and the Common Mission Definition proposals, these should be used to create the critical base fragment elements.

3.5.3 Fragment Schema Elements. It is desirable that changes to the base elements of these fragments are minimal and infrequent. When changes are needed, they should be made through the additions of *optional* elements whenever possible (to allow backward compatibility).

Another option considered in this research was to require that initial fragment schemas allow the use of a catch-all element for every fragment (which would allow any IO publisher to add an unnamed element to an instance document). Section 2.3.7 explained the use of for this purpose. However, xsd:any is not proposed here because it introduces additional complications in the storage of the objects when

المنارة للاستشارات
www.manaraa.com

using a relational database, which is the current storage method used. The primary problem in arises because there is no column created for an unplanned element at table creation time. As such, this catch-all element would be of limited use because it could not be made searchable because no column exists for this element in the IO table. That means the optional element can be made part of the (non-searchable) object payload. Subscribers could still receive and view it, but it is not a useful schema element if it cannot be searched with a predicate.

Optional elements have enough flexibility for many COIs to be able to use these fragments and still have enough freedom to modify imported fragments within their platforms. Versioning and coercion methods will handle the more complex changes.

3.5.4 Central Namespace for Fragments. As previously stated, a common object engineering methodology would be especially useful for JBI communities that want to conform to a standard to allow maximum visibility of their published objects. For this research, all fragments have been defined in a single namespace (using XML Schema include references). However, it is recommended that a central namespace for fragments be created that can be utilized by multiple platforms (using XML Schema import references). In Section 2.3.3, several hybrid approaches for using namespaces for different types of projects were discussed. Since the variety and scope of JBI platforms cannot be considered at this early stage of development, no single approach is recommended. However, the very notion of standardization suggests that very basic fragments can be defined in a central namespace, with great flexibility for extension and restriction of the base elements when the fragments are inherited. XML namespaces allow validation of a schema when there is a unique filename at a specified location. Due to this restriction, a schema or fragment filename must be unique within a location, so it is proposed that file names for fragment schemas append the version as part of the file name (as proposed in fragment naming conventions: *fragname_frag_majorVersion_minorVersion.xsd*)

www.manaraa.com

For this research, it is assumed that base fragments will have been imported from a central location into a fragment table within the MSR.

3.5.5 XML Inclusion to Build Schemas. After several fragments are defined, the next step was to build new IO schemas from these fragments. This requires an inclusion mechanism within the IO type schemas. Inclusion mechanisms allow included components to change, without having to change the format of the top level schema. The two different standards developed by the W3C Working group that could each handle the inclusion are discussed in Section 2.3.5. The selection of the XML Schema method (using include and import statements) was based on the evaluation of each option's benefits, ease of use, and flexibility. The biggest factors promoting the use of the XML Schema methods of include and import are that they are fully supported by most validators, specifically created for component schema development and require no preprocessing before validation.

Figure 3.2 shows a schema which uses the geospatial_fragType and the target_fragType as well as the base object fragment. It should be noted that the schema must both "include" (or "import") the applicable schema and then declare the elements which are to be of the complex types defined in the fragment schema file. "Include" is used in this example since all the schemas belong to the fictitious http://www.myjbinamespace.com. If the JBI platform is using validation, the namespace declaration should point to the actual location of these schemas. For this research, validation is assumed to have happened before publishing new IOs.

3.5.6 Fragment Tables. Two tables are needed for the management of fragment overhead, which have been added to the MSR as described in Table 3.1. These tables are being used both for fragment management (creating and storing new fragments with their associated schemas and updating existing fragment schemas) and to serve as an index for searching for all IO types containing a fragment. The sample fragment schemas and some fragment-IO type pairings for geospatial_frag and

المنارة للاستشارات
www.manaraa.com

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
targetNamespace="http://www.myjbinamespace.com" xmlns="http://www.myjbinamespace.com">
                <xsd:include schemaLocation="baseObject_1_0.xsd"/>
                <xsd:include schemaLocation="geospatial_frag_1_0.xsd"/>
                <xsd:include schemaLocation="target_frag_1_0.xsd"/>
<xsd:element name="metadata">
     <xsd:complexType>
          <xsd:sequence>
                <xsd:element name="baseObject" type="BaseObjectData"/>
                <xsd:element name="geospatialData" type="geospatial_fragType"/>
                <xsd:element name="targetData" type="target_fragType"/>
          </xsd:sequence>
     </xsd:complexType>
</xsd:element>
</xsd:schema>
```

Fragment Library Files

Fragment Declarations

Figure 3.2: Component Schema with includes

target_frag are shown in the new fragment tables in Figure 3.3. The create table statements needed to add the fragment tables to the new MSR are in Section B.1.

After new fragments are added to the MSR, they are added to the fragment table. When new IO type schemas (composed of fragments) are added to the MSR, a row is added to the fragment_io table for each fragment type-IO type pairing in the new schema. The primary use of the fragment table is fragment management only. An *ID* field is used as a primary key to uniquely identify specific fragments in this table. The ID field is also a foreign key and part of the primary key in the fragment_io table. The fragment_io table has a primary key composed of three fields, since it takes all of these (ID, io_type, io_type_version) to uniquely identify a row in that table.

The use of the ID column as an index in the fragment_io table speeds searching because the physical ordering of the records impacts how many hits are in cache as

www.manaraa.com

	ID	fragment_name	fragment_version	fragment_schema
1	4	geospatial_frag	1.0	<?xml version="1.0" encoding="UTF-8"?> <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"> <xsd:complexType name="geospatial_fragType"> <xsd:annotation> <xsd:documentation>geospatial fragment</xsd:documentation>
2	8	target_frag	1.0	<?xml version="1.0" encoding="UTF-8"?> <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"> <xsd:complexType name="target_fragType"> <xsd:annotation> <xsd:documentation>target fragment</xsd:documentation>
3	10	weapon_frag	1.0	<?xml version="1.0" encoding="UTF-8"?> <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"> <xsd:complexType name="weapon_fragType"> <xsd:annotation> <xsd:documentation>weapon fragment</xsd:documentation> </xsd:annotation>

(a) fragment Table

	ID	fragment_name_version	io_type	io_type_version
1	4	geospatial_frag_1_0	fragment.fragSchema0	1.0
2	8	target_frag_1_0	fragment.fragSchema01	1.0
3	4	geospatial_frag_1_0	fragment.fragSchema01	1.0
4	4	geospatial_frag_1_0	fragment.fragSchema012	1.0
5	8	target_frag_1_0	fragment.fragSchema012	1.0
6	10	weapon_frag_1_0	fragment.fragSchema012	1.0
7	4	geospatial_frag_1_0	fragment.fragSchema0123	1.0
8	2	ato_frag_1_0	fragment.fragSchema0123	1.0
9	10	weapon_frag_1_0	fragment.fragSchema0123	1.0
10	8	target_frag_1_0	fragment.fragSchema0123	1.0

(b) fragment_io Table

Figure 3.3: Fragment Tables

المنارة للاستشارات
www.manaraa.com

Table 3.1: New MSR Tables

Table Name	Description	Field types
fragment	fragment file names	Primary Key ID field (int), varchar fields for fragment name and version, and text field for full fragment schema
fragment_io	fragment-IO type pairings for new schemas	ID field (foreign key from fragment table), varchar fields for concatenated fragment name and version, IO type name and IO type version

a result of a SQL query (due to the clustering of the fragment types and the smaller storage requirements of an *int* data type versus a *varchar*). Since a fragment schema file name must be unique within a namespace (to prevent name collisions on differing versions of fragments), a fragment name/version field, *fragmentName_version*, is used to represent the filename in the fragment_io table. This also enables searches on a single column when querying by fragment (fragmentName_version rather than fragment name *and* fragment version.

The fragment_io table is updated as each new IO type schema is added to the MSR. Each referenced fragment schema in the IO type schema requires a separate entry in this table. An XML preprocessor can handle this overhead by searching for includes and imports (see next section). The base object fragment should automatically be stored in the fragment table when a new JBI is stood up, and every new IO type schema added to the MSR should have a row added for the IO type-base object pairing in the fragment_io table (in addition to all the other pairings added for this IO type).

3.5.7 Fragment Processing Techniques. Now that the MSR has two additional tables, there is additional preprocessing required to maintain the new fragment tables. The following are the types of processing actions that need to be performed and the actions that trigger them (some of these actions have been manually accomplished for this research, but are suggested as platform integration recommendations):

www.manaraa.com

- **Adding new fragment schemas:** A new row is added to the fragment table to store the fragment name and its entire schema. The version number will be appended to the name. Additional checks are needed to prevent duplicate fragments.
- **Adding new IO type schemas:** A new row is added for each IO type-fragment type pairing for each fragment included in the new schema. Pairings will include version types as shown in the fragment_io table in Figure 3.3.
- **Updates or version changes to fragment schemas:** When a fragment schema version change is made, a new fragment entry is added to the fragment table. If the update does not warrant a version number change as defined in Section 3.5.8, only the schema is updated. In this research, all of the sample fragments are of the same version number.
- **Updates or version changes to IO type schemas:** When an IO type schema version change is made, new rows must be added to the fragment_io table with the new fragment-IO type pairings.

3.5.8 Versioning Standards. Since this research proposes to have all schemas composed of one or more fragments, IO type schema version changes should be warranted for either (or both) of only two reasons:

- **Changes to fragments:** There may be changes to one or more fragment schemas that are already present in an IO type schema. These should be classified and implemented as either a major or minor fragment schema change and should prompt the same change type to all of the IO type schemas which contain this fragment schema.
- **Adding/Deleting entire fragments:** A new fragment schema may need to be included in an existing IO type schema. While this is not a change to any fragment schema, it warrants a change to the IO type schema.

www.manaraa.com

Fragment schema version changes will be classified based on an evaluation of whether they allow forward and backward compatibility. A schema change allows compatibility if a schema instance would pass validation checks against both an old and new version. Platform actions to enable publish and subscribe to continue seamlessly after version updates are addressed in the next section. Given this reasoning, the following are the proposed rules for fragment schema versioning:

- **Minor change:** Example: Version 1.0 to Version 1.1. A fragment schema change shall be classified as minor if there is an addition or deletion of any *optional* XML metadata element or attribute anywhere in the schema. This is the only valid minor change because instances of both old and new schema versions will pass validation on either the old or new schemas. As previously stated, a minor version fragment schema change will promote a minor version IO type schema change of all the IO types containing that fragment.
- **Major change:** Example: Version 1.0 to Version 2.0. A change shall be classified as major if it cannot be classified as minor. These kinds of changes will include data type changes, element and attribute name changes, addition (extension) or deletion (restriction) of non-optional attributes and elements, and cardinality changes (including changing an optional element to mandatory, since this is a change to the "minOccurs" attribute). As previously stated, a major version fragment schema change will prompt a major version IO type schema change of all the IO types containing that fragment. It was also noted that an IO type schema can undergo a change if a fragment was added, and this change to the IO type will also be major.

3.5.9 Changes to Pub/Sub/Query. Major version changes will mandate changes to publish, subscribe and query actions. These changes must be made by the platform (through coercion instructions as proposed in Section 3.5.10) and also by clients (by changing applications which publish, subscribe or query for information objects). To prevent unanticipated disruption of critical services, version changes

المنارة للاستشارات
www.manaraa.com

should only be done on a scheduled basis. A configuration management table can manage change submissions until the update is processed. Notification should be made to subscribing and publishing clients of the fragments and/or IO type changes. This should be done both prior to and after the changes have been made to ensure action will be prompted by clients. The platform broker which manages subscriptions and publications should make this notification to all clients in its registry. Any new clients should only be permitted to establish subscriptions and publications using the new types.

When a version change has been made to an IO type, a new table will be created in the Information Object Repository (IOR) to store IOs of the new type. New fields will be added or deleted, as appropriate, and a new field for version reference will be added. This field will keep track of the actual version number of that particular object, as published. When subscriptions are filled, coercion translation will ensure subscribers receive the versions they subscribe to, as the published type will be translated to the subscribed type (if allowed). Elements of published IOs of the old version will be translated to conform to the new version. The previous version number will be stored in the version reference field. Subscriptions and queries of this object will be able to retrieve IOs of old and new versions from the single table, because coercion will perform the necessary translations, as described in the next section.

By allowing old publications and subscriptions to continue, there will be less disruption to subscriber applications than if subscriptions were abruptly upgraded when new IO type versions are available. Client applications may not be able to process new IOs when there are IO type changes and it may be costly in terms of time, effort and/or system disruption to require these subscribers to immediately conform to new IO types. Clients should be encouraged to migrate their applications, but can then do so on their own timeline. The coercion mechanism is critical to allowing updates to process seamlessly from the client's point of view.

المنارة للاستشارات
www.manaraa.com

3.5.10 Coercion Techniques. A coercion table will become part of the MSR. This table will record platform or client actions required to translate IO types to older versions to fulfill active subscription of these older IO types. For each of the element and attribute changes made to an IO type schema, there will be one instruction. Only one coercion entry should be necessary per element or attribute change (as opposed to one for the fragment types and one for the IO types).

There will be no instance of fragment coercion that takes place outside the context of an IO type coercion. For example, a fragment schema may have a version change before it has ever been used in an IO type schema. No coercion is needed in this case because there are no IO types to coerce and clients should be forced to only use the new fragment schema version in new IO type schemas.

The coercion table will have one row for each type of change to a fragment within an IO type. The table will provide a map for the platform to convert old objects versions to newer versions *and* to enable backward compatibility to deliver objects to subscribers of old IO type versions. The proposed structure of this table is shown in Table 3.2.

Most column descriptions in this table are sufficient to describe the contents. Others require some elaboration. Coercion rules with the same "From_Version" and "To_Version" IO type values should be numbered incrementally and processed in the same sequence so as to preserve the order of element node mappings and possible multiple changes to the same element.

Element_Node refers to the node that this coercion rule applies to in the IO type schema. This is based on a tree node mapping of the old version of the IO type schema. This reference is to the applicable fragment in the IO type schema after all previous coercion rules have been processed. For example, refer to Figure 1(a). In the geospatial_fragType shown, there are 10 parallel elements. The "humidity" element is the 10th parallel node within this complex type. Thus, this element node number in the component schema would be 10. If this schema had contained any nested

المنارة للاستشارات
www.manaraa.com

Table 3.2: Coercion Table Structure

Field Name	Description
IO_Type	Name of IO type changed
From_Version	Old IO type version
To_Version	New IO type version
Fragment_Name	Name of fragment changed
Frag_From_Version	Old fragment version
Frag_To_Version	New fragment version
Type_change	This change type (e.g., add element, delete element)
Element_Node	Metadata node address at which to add, delete, or make change
Change_From	From value (if applicable)
Change_To	To value (if applicable)
Data_Type	New data type (if an element change or addition, restriction, or extension)
Restriction_Base	Base data type (if adding or removing restriction to an element)
Restriction_Name	Restriction Attribute name (if adding or removing restriction to an element)
Extension_Base	Base data type (if extending or removing extension of an element)
Extension_Name	Extension Attribute name (if extending or removing extension of an element)
Default_Value	Default value for new/changed field, restriction, or extension (if non-null and value needed)
Coercion_Rule	Platform instruction to handle conversion to the new version

element sequences within an element, they would be represented as ParentNodeNumber.ChildNodeNumber.ChildNodeNumber, etc. For example, 2.3.7 represents the 7th element of the 3rd element of the 2nd element. When a coercion rule calls for a node deletion, it refers to the node location in the old version. When a coercion rule calls for a node addition, it refers to adding the node after the referenced node (thus, giving the *added* element that node number + 1).

The Change_From and Change_To can have different meanings depending on the type of change. Table 3.3 provides the descriptions of the field contents for these columns for the corresponding type of change. Many columns will only contain data

www.manaraa.com

if it pertains to that type of change. For example, Restriction_Name will only contain data if this change will be adding or removing restriction to an element.

Coercion_Rule will contain any additional programming instructions that may be needed for the platform to process the translation for this element.

Client-side coercion should also be offered by platforms. In this case, clients are provided the coercion table instructions for IO types to which they subscribe. This may be more desirable in cases where the overhead for platform coercion is too costly and/or not needed. For example, a client has evaluated the coercion instructions for IO type "A" version 1.0 to 2.0. They have determined that the object processing application for this subscription does not evaluate or process any of the metadata fields which require coercion. Furthermore, they are processing critical battlefield statistics for which the speed of receiving and processing the objects is paramount. In this case, coercion is not warranted and potentially detrimental in terms of execution time. Thus, coercion options should be available to clients to either accept or refuse on queries and subscriptions.

A coercion example is provided in Figure 3.4. In this example, *geospatial_frag* requires coercion from version 1.0 to 2.0 in IO type *A*. The *Coercion Table* (Figure 4(b)) shows 4 changes as highlighted in Figure 4(a), which have sequence numbers from 1 through 4. As previously stated, coercion must follow the sequence order. The element *regionID* must first be added after node 10 as a string data type. This element is optional and thus requires an enumeration attribute of *minOccurs="0"* at node 11, which is now the number of the inserted node. This illustrates why changes must follow the proper sequence. The final two changes are data type changes on two elements from integers to decimals. Although the new element is optional and would have only necessitated a minor version change, the data type changes require a major change according to the prescribed rules. As is required, the fragment version change triggered the same grade of schema version change to IO type *A*.

المنارة للاستشارات
www.manaraa.com

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
      <xsd:complexType name="geospatial_fragType">
            <xsd:annotation>
                  <xsd:documentation>geospatial fragment 1.0</xsd:documentation>
            </xsd:annotation>
            <xsd:sequence>
                  <xsd:element name="MajorVersion" type="xsd:integer"/>
                  <xsd:element name="MinorVersion" type="xsd:integer"/>
                  <xsd:element name="lowLat" type="xsd:decimal"/>
                  <xsd:element name="highLat" type="xsd:decimal"/>
                  <xsd:element name="lowLong" type="xsd:decimal"/>
                  <xsd:element name="highLong" type="xsd:decimal"/>
                  <xsd:element name="lowElevation" type="xsd:integer"/>
                  <xsd:element name="highElevation" type="xsd:integer"/>
                  <xsd:element name="visibility" type="xsd:decimal" minOccurs="0"/>
                  <xsd:element name="humidity" type="xsd:decimal" minOccurs="0"/>
            </xsd:sequence>
      </xsd:complexType>
</xsd:schema>
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
      <xsd:complexType name="geospatial_fragType">
            <xsd:annotation>
                  <xsd:documentation>geospatial fragment 2.0</xsd:documentation>
            </xsd:annotation>
            <xsd:sequence>
                  <xsd:element name="MajorVersion" type="xsd:integer"/>
                  <xsd:element name="MinorVersion" type="xsd:integer"/>
                  <xsd:element name="lowLat" type="xsd:decimal"/>
                  <xsd:element name="highLat" type="xsd:decimal"/>
                  <xsd:element name="lowLong" type="xsd:decimal"/>
                  <xsd:element name="highLong" type="xsd:decimal"/>
                  <xsd:element name="lowElevation" type="xsd:decimal"/>
                  <xsd:element name="highElevation" type="xsd:decimal"/>
                  <xsd:element name="visibility" type="xsd:decimal" minOccurs="0"/>
                  <xsd:element name="humidity" type="xsd:decimal" minOccurs="0"/>
                  <xsd:element name="regionID" type="xsd:string" minOccurs="0"/>
            </xsd:sequence>
      </xsd:complexType>
</xsd:schema>
```

(a) Fragment Files

COERCION TABLE				
Coercion Sequence	A_1.0_2.0:1	A_1.0_2.0:2	A_1.0_2.0:3	A_1.0_2.0:4
IO Type	A	A	A	A
From Version	1.0	1.0	1.0	1.0
To Version	2.0	2.0	2.0	2.0
Fragment Name	geospatial_frag	geospatial_frag	geospatial_frag	geospatial_frag
Frag From Version	1.0	1.0	1.0	1.0
Frag To Version	2.0	2.0	2.0	2.0
Type change	Element Addition	Element Enumeration	Element Data Type	Element Data Type
Element Node	10	11	7	8
Change From	null	minOccurs	null	null
Change To	regionID	"0"	xsd:decimal	xsd:decimal
Data Type	xsd:string	null	xsd:decimal	xsd:decimal
Restriction Base	null	null	null	null
Restriction Name	null	null	null	null
Extension Base	null	null	null	null
Extension Name	null	null	null	null
Default Value	null	null	null	null
Coercion Rule	Insert default value in new column	None (makes this element optional)	Cast integer value to decimal	Cast integer value to decimal

(b) Coercion Table

Figure 3.4: Coercion Example (geospatial_frag: version 1.0 to 2.0)

www.manaraa.com

Table 3.3: Coercion Table Change Descriptions

Type_change	Change_From	Change_To
Element Addition	null	New element name
Element Deletion	Old element name	null
Element Data Type	null	New data type of added or changed element
Attribute Addition	null	New attribute name
Attribute Deletion	Old attribute name	null
Attribute Data Type	null	New data type of added or changed attribute
Element Enumeration	Enumeration constraint	New constraint value or null for constraint removal
Element Add Restriction	null	New restriction value
Element Remove Restriction	Old restriction value	null
Element Change Restriction	Old restriction value	New restriction value
Element Add Extension	null	New extension value
Element Remove Extension	Old extension value	null
Element Change Extension	Old extension value	New extension value

3.5.11 Information Object Storage Modifications. Each IO type has its own relational database table within the IOR. IOs are stored in a table by hashing each relative metadata tag path within the XML document to a unique ID that is then stored in its own row with the value of that tag in the same row. As discussed in Section 2.2.3.1, this method of storage has several drawbacks, the most significant of which is that this hash technique does not allow repeated metadata elements (each path must be unique). The repeated element values are not "lost" from the information object (that is to say, they will be returned on a query of subscription with the rest of the IO). However, only the last element of a repeated list will be "visible" for predicate matching.

Another drawback of this hash technique is that it does not allow XML attribute matching. Only XML elements can be used to build predicates in the current JBI RI. This may not be particularly detrimental because most attributes can also be represented as elements. The AFRL JBI branch was exploring the use of the open source database dbXML (Native XML Database) and other solutions to solve the repeated

المنارة للاستشارات
www.manaraa.com

namespace problem. However, no IO storage modifications have been required for this research. This proposal should have merit regardless of which storage mechanism is used. The benefits of fragment inclusion is related to the MSR (not the IOR) through the implementation of fragment indexing and suggested CAPI modifications.

3.5.12 Common API Changes. Several CAPI methods are required to allow manipulation of fragments within the platform. The list of methods and the Structured Query Language (SQL) queries used to implement these methods are shown below. This list is not all inclusive, but includes the critical methods for adding and retrieving information from the fragment tables.

Add Fragment to fragment Table:

METHOD:

boolean addFragment (String name, String version, String schema)

SQL STATEMENT:

INSERT INTO fragment (fragment_name, version, fragment_schema)
VALUES (‘fragName_frag’, ‘1.0’, FULL_SCHEMA(Note 1))

Add Fragment-IO type pair to fragment_io Table:

METHOD:

boolean addFragmentIOType (String ioType, String IOTypeVersion, String fragmentName, String fragmentVersion)

SQL STATEMENTS:

1) SELECT ID as fragmentID FROM fragment WHERE
fragment_name = fragmentName and fragment_version = fragmentVersion

2) INSERT INTO fragment_io (ID, fragment_name_version, io_type, io_type_version)
VALUES (fragmentID, fragmentName, IOType, IOTypeVersion)

Get an array of IO Types for a given set of fragments:

METHOD:

المنارة للاستشارات
www.manaraa.com

Vector getInfoObjectVectorByFragment (String[ ] fragmentArray)

SQL STATEMENT(Note 2):

SELECT io_type, io_type_version, count(*)

FROM fragment_io WHERE fragment_name_version = 'frag1_1_0' OR

fragment_name_version = 'frag2_1_0' OR fragment_name_version = 'frag3_1_0'

GROUP BY io_type, io_type_version HAVING COUNT(*) = fragmentArray.length

Get an array of fragments for a given set of IO Types:

METHOD:

Vector getFragmentVectorByInfoObject (String[ ] IOTypes)

SQL STATEMENT(Note 3):

SELECT DISTINCT fragment_name_version

FROM fragment_io WHERE

io_type = 'IOtype1' OR io_type = 'IOtype2' OR io_type = 'IOtype3'

(Note 1) FULL_SCHEMA refers to a string representation of the full length schema file for this fragment.

(Note 2) The COUNT(*) = fragment_array.length and GROUP BY clause limit the results to IO types containing *all* of the fragments in the given array.

(Note 3) The DISTINCT keyword eliminates repeat fragment names for those fragments that may be present in more than one IO type in the SQL statement.

3.6 Environmental Parameters

This research has been implemented on a JBI configuration where the server, client and database repository all reside on the same hardware. The proposed modifications to realize the IO engineering methodology have been implemented with an application built outside the platform and manual changes and additions to the MSR. To ensure accuracy of evaluation of both the current and proposed methods, both were tested outside the platform, because platform processing overhead could

المنارة للاستشارات
www.manaraa.com

impact negatively on the current method and skew the results. To fully realize the implementation of this proposal, the proposed changes should be integrated with the JBI core services in a future release.

3.7 Evaluation

There are several ways to evaluate whether this proposed methodology is an improvement to the current system, although not all of these are quantifiable. For example, a specific IO engineering methodology has not been proposed for this system before this research. Consequently, there is no means for comparison of this methodology other than that it proposes something standardized, distributable and reusable.

There is also no means by which to completely measure the impact on the performance of the JBI core services for subscriptions and queries because the overhead is being measured on the side of the application that has been developed, which is outside the platform. What *can* been measured is the time it takes to obtain the same result set for both systems.

3.8 Hypothesis and Interpretation of Results

It is apparent that a multi-object search method will be an improvement over the current method of a single object query or subscription. However, the greater savings are the reduction in time to find all the IO types with identical metadata fragments. The only way to do this with the current system is through an iterative search through IO types. The proposed fragment_io table is used to index IO types by fragment, so the time savings should be exponential.

3.9 Summary

The primary goal of this research is to introduce a Fragment-Based Information Object Engineering methodology and provide standards for the integration of this into the architecture of the JBI platform. The greatest improvement that will be realized

www.manaraa.com

by implementing this proposal is that there will be a defined standard where none has previously existed. The research includes proposed changes to the MSR, the addition of CAPI methods to retrieve related IO types and fragment types, and definitions and methods for versioning and coercion. The methodology included a fragment definition, proposed revisions for the MSR to store and index the fragments, and provided standards for fragment and IO type revisions.

Chapter 4 covers the implementation details while Chapter 5 discusses the analysis of the results of this research. Chapter 6 provides a summary and conclusion as well as recommendations for future study and platform integration.

المنارة للاستشارات
www.manaraa.com

IV. Implementation

4.1 Introduction

This chapter covers the system implementation of the Fragment-Based Information Engineering methodology. To test this proposal, several tasks had to be accomplished before evaluation. Most of the preparation involved installing and populating a sample JBI platform and data repository which is discussed in *Preliminary Operations*. Then, an application was written to compare the retrieval performance for both the current and proposed methods on the same data sets. This is discussed in *Test Application*.

Some aspects of the proposal are offered without test or implementation. The reason for including them is that they may provide a great enhancement to the architecture. However, it was not possible to integrate them into the system (and thus be evaluated), given the research time constraints. In other cases, there was a lack of a core functionality available in the latest version of the JBI Core Services platform. For example, coercion techniques were introduced here even though this capability is not yet part of the JBI Reference Implementation (RI) Version 1.2 (the latest version of the core services at the time of this research).

4.2 Compromises

Some compromises were required to conduct this research. For instance, improvements are proposed to both the storage methods and retrieval mechanisms. These improvements require substantial platform modifications to fully implement and test the solutions inside a JBI platform. As such, some improvements were tested *outside* the platform. This means that the database repository was manually changed and then manipulated with an application that did not utilize the platform core services or Common API methods.

The current platform also had limited functionality to perform actions similar to the proposed methods. In some cases, the platform also had overhead associated with the current methods that did not exist in the proposed methods. In most cases, the

المنارة للاستشارات
www.manaraa.com

underlying web service actions were responsible for this overhead. For these reasons, removing the platform actions provided an opportunity for more accurate evaluations of current versus proposed methods.

4.3 Preliminary Operations

There were many preliminary operations that had to be accomplished before any testing could begin. The first order of business was to install and configure the JBI Core Services and other required programs on a test machine. These specifications and configurations are discussed in detail in Chapter 5. Other required operations were related to choosing sample sizes and creating and storing fragments, object schemas, and object instances.

Before adding actual test data to the Metadata Storage Repository (MSR), several sample Information Object (IO) type schemas were created and added to help determine the platform actions and requirements for adding IO type schemas. This also enabled evaluation and speculation of possible database and platform limitations for this research.

4.3.1 Fragment Library. To help select a size for the initial fragment library, a decision was made that the size of the library would determine the number of IO type schemas stored in the MSR. Every possible combination of fragments would be used to create IO type schemas. The number of schemas for these combinations for n fragments is given by Equation 4.1.

$$\sum_{i=1}^{n}(n)C(i) - 1 = 2^n - 1 \tag{4.1}$$

Thus, the number of possible schema combinations would approximately double for each fragment added to the library. While this exhaustive fragment use assumption is likely not a realistic comparison to an actual JBI repository, it was useful for ensuring predictable numbers of matching IO types when executing tests. To aid further in

المنارة للاستشارات
www.manaraa.com

the choice of n, it was decided that the same number of object instances would be published per IO type.

The most limiting factor for the choice of n was the database storage requirements. The architecture of the JBI Information Object Repository (IOR) requires one table for every schema. It was speculative (without a deployed JBI instance), but it is more likely that there would be a limited number of IO types with many IOs published per type than many IO types with a limited number of IOs per type. Even if the entire contents of the Department of Defense Metadata Registry (DODMDR - refer to Section 2.4) were used, there are only 2300 total schemas in that entire registry. A single JBI community repository should only represent a small fraction of the entire DOD information landscape. Also, there was no real world set of fragments (of a small enough scale) discovered early enough in this research effort to be useful for testing. The Common Mission Definition (See Section 2.4) would have been very useful but the working draft was only recently acquired. Recommendations for the integration of these data components into an initial fragment library are described in Chapter 6.

Since these elements were not available at These considerations led to a selection of n=10, and the decision to create a simple set of basic fragments that would be just sufficient to demonstrate the methodology. The resulting number of IO type schemas is then given by Equation 4.2

$$2^n - 1 = 2^{10} - 1 = 1023 \tag{4.2}$$

With an arbitrary selection of 500 objects per schema, this would result in approximately 500,000 stored objects. Since the goal of this research was not to test the storage limitations of the database, this was sufficient to test the viability of the proposed method.

المنارة للاستشارات
www.manaraa.com

Table 4.1: Fragment Schema Distribution

Number of Fragments	Number of IO Types
1	512
2	256
3	128
4	64
5	32
6	16
7	8
8	4
9	2
10	1

For 1 to n, Equation 4.3 determined the number of IO type schemas which had the given number of fragment combinations in its schema. Table 4.1 shows the IO type schema distribution by number of fragments in the schema for 1 through n.

$$2^{(n-1)} \tag{4.3}$$

4.3.2 Platform Limitations. The first sample IO type schemas were added through the platform interface. This was done so that the database could be analyzed to determine what actions were performed when a new IO type was added to the MSR, and also to see how prohibitive in terms of time it would be to add more than 1000 schemas one at a time. Each of these IO type schemas took less than a minute to add through the platform interface. However, as the number of stored schemas grew, the platform delays grew much longer (because the platform had to update an increasingly larger IO type schema package tree). This reinforced the decision that it would be necessary to manually add the schemas to the repository. As such, an examination was made of the platform actions taken when a new schema was added. The actions prompted by this study are discussed in the next section.

www.manaraa.com

4.3.3 Populating Database Tables. The database study revealed that there were three basic database actions performed by the platform when any new schema was added:

- Insert the schema and other related data into the database table: ***mdr.ior_repository***. The name of this table is a misnomer because it resides in the MSR (which stores IO type *schemas*), instead of the IOR (which stores *information objects* (IOs)).
- Insert security permissions data into the database table: ***security.privilege_store***.
- Create a new table for the IOs in the ***ior*** database with the unique table name: ***IOTypeName_majorVersion_minorVersion***

Each of these actions were examined in detail. The schema table was a simple structure with one row per schema with very descriptive field names. Thus, updating this table required a single SQL statement to insert a row into the table for each schema. The structure of this table was introduced in Figure 2.3. Specific programs written for creating and populating tables are covered in Section 4.3.8.

The other two required actions required a bit more examination to determine what would need to be added to the database. These evaluations are discussed in the next two sections.

4.3.4 Permissions. The examination of the security permissions table ***security.privilege_store*** revealed that this security table required 18 tuples per IO type for a sample user. The large number of rows needed per schema were required to grant a user full permissions to add, delete and update schemas, and publish, delete and subscribe to objects. This large number of security rows meant that for 1023 schemas, the size of this table would be greater than 18000 rows for this test.

4.3.5 IOR Table Fields. The storage format for IOs in the JBI IOR was discussed in Section 2.2.3.1. The hash methods for generating field names was not

المنارة للاستشارات
www.manaraa.com

Table 4.2: Fragment Names

Fragment Number	Fragment Name
0	geospatial_frag
1	target_frag
2	weapon_frag
3	ato_frag
4	aircraft_frag
5	personnel_frag
6	bldg_frag
7	sensor_frag
8	vehicle_frag
9	report_frag

available, so steps had to be taken to decipher the actual mapping for the metadata element nodes in each of the fragments. This was accomplished by adding each single fragment IO type schema to the MSR through the JBI-provided interface, then publishing some sample IOs to the IOR. This technique enabled a reverse mapping of table field names to IO type schema elements. This mapping was also accomplished for each of the base object fields which were also required by the platform. If these base object fields were not added, the platform would not properly recognize and store the IOs when they were published.

After the mapping of every column was done, a file containing create table statements for each of the schemas was created. The program utilizing these create table statements is covered in Section 4.3.8.

4.3.6 Combination Generator. Fragments were given descriptive names and were associated with a number as shown in Table 4.2. A naming convention was chosen for IO type schemas that would allow a mathematical algorithm to be used to generate all schemas, instances and file names from the fragment numbers.

A combination generator program was used to create all possible fragment combinations of the fragment based IO type schemas. This program was used as a helper class for the java programs which created the many IO type schema and IO instance

www.manaraa.com

files, populated the MSR and security tables, created IO storage tables, and ran a publishing sequence for the sample objects.

The CombinationGenerator.java class was obtained, which used a program derived from a discrete mathematics algorithm for generating every possible combinations (in-order) of a given string [14, 23]. The java source code for this program is in Section B.2.

Using a string array of the assigned fragment numbers (0 to 9), and the naming convention defined in the next section, the generator enabled the assignment of unique and descriptive IO type names, unique file names, proper file contents for schemas and instances, and proper arguments for create, insert, and update SQL statements.

4.3.7 Creating Fragment Schemas and Instances. Each of the fragments was assigned from 3 to 10 metadata elements. These elements were assigned sample values for the instance documents. Each combination of fragments would have only one of five possible object instance configurations for published objects. Fragment metadata elements, their mapped hashed field names, and the five possible instance combinations are shown in Figure A.1.

For simplicity, every schema using fragments was assigned to the same package (*fragment*) and had the same name prefix (*fragSchema*). The number combination appended to the name prefix matched the in-order number combination of the fragments present in this schema. For example, the schema containing the geospatial, target and weapon fragments was named *fragSchema012*. The combination generator and a switch/case statement with a case for each number string was used to determine file names, file contents and SQL statements.

Every schema first contained the base object data (as required by the platform), and then the fragments in numerical order. Fragment schemas were also not assigned to any actual XML namespace. The schemas were originally assigned to a fictional namespace, but the platform would not properly support either a schema or instance containing *any* namespace declaration except *xsi:noNamespaceSchemaLocation* in the

المنارة للاستشارات
www.manaraa.com

instance documents. This is because namespaces are not yet recognized by the JBI Reference Implementation (RI).

Since the platform also did not support the include or import schema declarations, a separate MSR (*frag.ior_repository* table) was created to exactly mirror the current MSR, with the exception that the schemas would only contain references to fragments (using include and import) rather than the full fragment declarations. This table was included in the database "frag", which contained the other fragment tables. The storage modification did not add any proposed functionality, but the reduced storage size per schema and referential integrity benefits have been evaluated in the analysis of the benefits of this new storage format in Chapter 5.

4.3.8 Creating and Populating Files and Tables. Using the aforementioned numbering and naming conventions, several small java programs were written to create the .xsd schema files and .xml object instance files, using the Combination Generator helper class and a file writer method. The programs and their function are listed in Table 4.3. Where the **Database.Table** column data is prefixed with a "FOR", that indicates the program generated a file that was then used by another program to insert the file data into that database table. There are also five instance file generators for the five different fragment instance values per fragments (as shown in Figure A.1). Finally, some update programs were necessary when it was discovered that a new column was needed or the data in a column would be more suitable in a different format. These programs are also included.

4.4 Test Application

A java application was written to perform a comparison of IO type retrieval times for the current and proposed MSR formats. The comparison is based on measuring the execution time to retrieve the same data using the techniques available for each configuration. The following sections discuss the test parameters and methods used to perform the evaluation.

المنارة للاستشارات
www.manaraa.com

Table 4.3: Preprocessing Programs

PROGRAM	ACTION	DATABASE.TABLE
CreateIOTables.java	Create one table for IOs of each IO type schema	ior
FragmentInstanceGeneratorTemplate	Create blank instance of each IO type schema	FOR mdr.ior_repository
FragmentInstanceGenerator.java	Create a file with an instance of each IO type schema	FOR ior.ior_repository
FragmentInstanceGeneratorA.java	Create a file with an XML instance of each IO type schema	FOR ior.ior_repository
FragmentInstanceGeneratorB.java	Create a file with an instance of each IO type schema	FOR ior.ior_repository
FragmentInstanceGeneratorC.java	Create a file with an instance of each IO type schema	FOR ior.ior_repository
FragmentInstanceGeneratorD.java	Create a file with an instance of each IO type schema	FOR ior.ior_repository
FullSchemaGenerator.java	Create a file with the full IO type schemas (including fragment definitions)	FOR mdr.ior_repository
SchemaGeneratorIncludes.java	Create a file with the IO type schemas (composed of fragment includes)	FOR frag.ior_repository
PopulateMSROld.java	Insert schema data into current MSR	mdr.ior_repository
PopulateMSRNew.java	Insert schema data into proposed MSR	frag.ior_repository
PopulatePrivileges.java	Insert 18 rows of security data per IO type schema	security.privilege_store
PopulateTables.java	Insert one row per fragment-IO type pair for each IO type schema	frag.fragment_io
UpdateFragMSR.java	Insert blank instance into new SCHEMA INSTANCE column for each IO type schema	frag.ior_repository
UpdateSchemas.java	Update SCHEMA column data for each IO type schema with corrected schema file	mdr.ior_repository

4.4.1 Evaluation Parameters. To compare the current IO type storage architecture to the proposed fragment-based method, the parameters were as follows. For the proposed (fragment-based MSR) system:

- The MSR has a library of n fragments.
- There is a theoretical maximum of $2^n - 1$ possible fragment combinations (schemas) from these fragments (all schemas are only composed of fragments and all possible combinations are represented as IO types in the MSR), so there are $2^n - 1$ IO type schemas in the MSR.
- Each fragment is present in 2^{n-1} of the schemas.

For the current (non fragment-based MSR) system:

المنارة للاستشارات
www.manaraa.com

- The MSR contains no fragment library but the platform in which the user is operating maintains a directory of IO types, which are of the same format of those that have been proposed in the fragment system.
- There are $2^n - 1$ IO type schemas in the MSR

For both the current and proposed systems, the user is assumed to have the same information needs to retrieve IOs (with the same predicates) and that other than some metadata element standardization, the user does not know exactly which IO types may contain these elements.

4.4.2 Current MSR Evaluation Modifications. When selecting an iterative search method for the current MSR, it was discovered that there was no database field in the current MSR for validating a relative XML metadata path against an instance schema without complex parsing of the stored XML schema (because the XPath predicates are *instances* and the database contains a stored *schema*). For this reason, a blank XML instance document was created for every IO type schema and inserted into the current MSR. This instance is not an object per se, because there were no values assigned to any of the metadata elements. Furthermore, this instance would be stored in a single table field for rapid retrieval and evaluation of relative metadata paths. This would allow the most rapid evaluation for matching schemas during the iterative search. Accordingly, this would ensure that the only execution time for the current MSR search was due to the actual cost of the iteration rather than the additional time for parsing the schema prior to the evaluation. The name of the new column added to the mdr.ior_repository table was schema_instance.

This new schema instance field is not a proposed integration to the MSR, as it was just needed for comparing the current architecture against the fragment solution. However, it has been considered as a possible improvement to the fragment table structure as discussed in the analysis in Chapter 5.

المنارة للاستشارات
www.manaraa.com

4.4.3 Current MSR IO Type Search. For the *current* system, for each array of 1 to n fragments, the application conducted a search on each IO type to see whether it contained (at a minimum) the fragments of interest. This resulted in an iteration through all 1023 IO types. The current Common API has no method to match a metadata path of an instance document to a stored IO type schema. Consequently, this search required the "blank" XML instance templates discussed in Section 4.3.7 to be used for relative XML path comparisons.

There were two options for performing the iterative search of the current MSR. The first option was:

- For each type in an array of all the IO type names in the MSR, perform a SQL query to retrieve its schema instance,
- Store the schema instance in a local string variable,
- Compare the fragment predicate to the instance string,
- Save matching IO type name to an IO type array to return to the calling function after completing the iteration of all IO types.

The second option was:

- Perform one SQL query to retrieve all schema instances and IO type names in the MSR,
- For each instance in the result set, store it in a local string variable,
- Compare the fragment predicate to the instance,
- Save matching IO types to an IO type array to return to the calling function after completing the iteration of the entire result set.

The choice between these methods represents a trade-off of SQL query execution time for local program memory consumption for the large array of IO types. Both of these techniques were used and evaluated in the analysis of this implementation.

المنارة للاستشارات
www.manaraa.com

For both search methods, the comparison was done by using the same sample XPath, composed of selections from each fragment's metadata "and-ed" together to form the predicate. As matching IO types were found, they were appended to an IO type vector that was returned to the calling function.

The method used to perform the multiple SQL query iterative search, *Vector getInfoObjectVectorBySearch(String[] allIOTypes, String predicate)*, is shown in Section B.3. The method used to perform the single SQL query iterative search, *Vector getInfoObjectVectorBySearch2(String predicate)*, is shown in Section B.4. Both methods used the XPath schema validator class detailed in the next section.

4.4.4 XML Instance to Schema Validator. The XPathEvaluator.java class contains the method used to do the node matching test on the XPath predicate. This source code was provided by the Air Force Research Lab (AFRL) in-house JBI Team and is shown in Section B.5. The method *boolean evaluate (String predicate, String metadata)* compares the predicate against a metadata instance string and returns true if there is at least one match on every relative path in the predicate to the instance. This method is able to process multiple paths in the predicate string (as in a case with multiple fragment paths "and-ed" together). The arguments required for this method are the predicate (fixed by fragment) and the IO type schema instance.

4.4.5 Proposed MSR IO Type Search. For the *proposed* system, for each array composed of 1 to n fragments, the application conducted only one SQL query using the fragment_io table as an index. The method used to retrieve the result set containing all matching IO Types from the fragment_io table, *Vector getInfoObjectVectorByFragment(String[] fragmentArray)*, is shown in SectionB.6. This method uses a single SQL statement to obtain a reference to all of the corresponding IO types. The calling function must only provide a string array of fragment names.

4.4.6 Application Interface. The test application interface is shown in Figure 4.1. The number of fragments and the number of iterations could be varied

المنارة للاستشارات
www.manaraa.com

between one and ten each time the test was executed. The application either tested the current or proposed method for any particular execution sequence. The source code was modified as needed to change the method which would be called for the two types of current MSR searches (multiple SQL or single SQL).

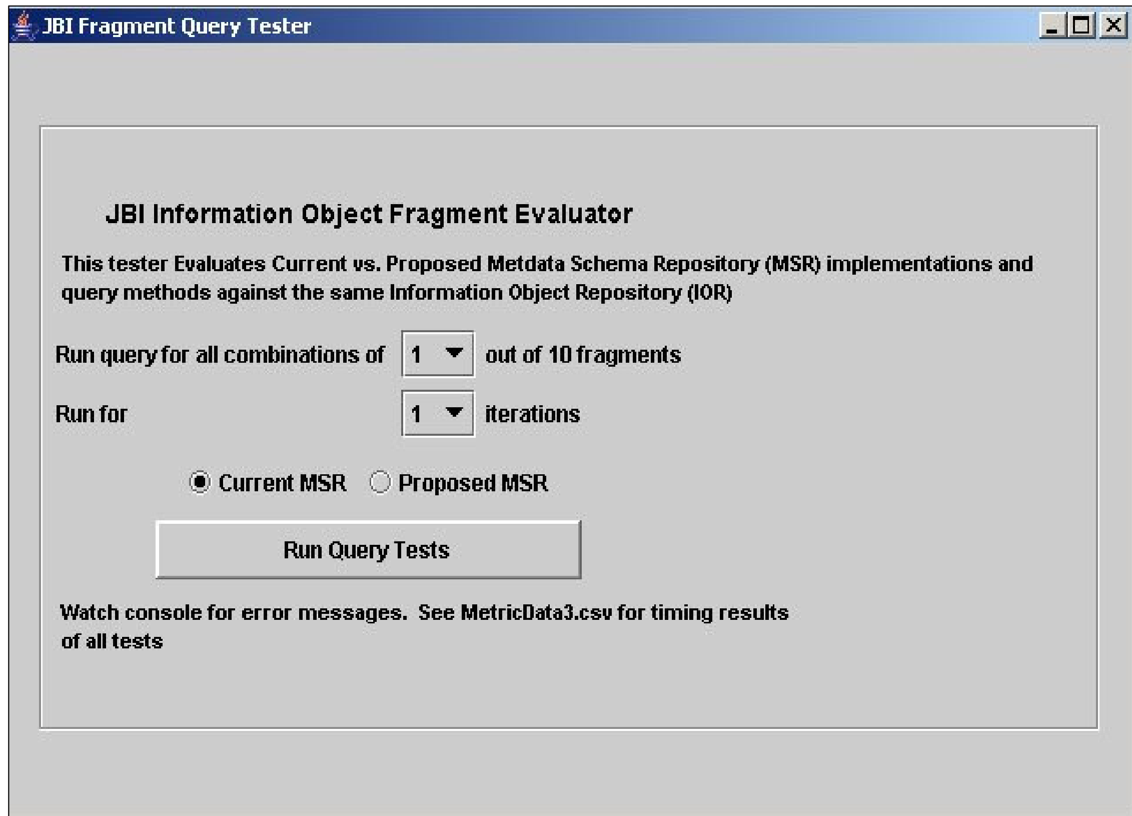


Figure 4.1: JBI Fragment Query Tester

4.4.7 Evaluation Measurements. Initially the application was configured with methods to retrieve the IOs using fixed query sequences and predicates after determining the proper IO types using either the iterative search or index method. This was later amended to just retrieve the *reference* vector of the IO types. The reasons for this change relate to platform processing overhead and common retrieval methods as explained in this section.

www.manaraa.com

There is a 10 step process to retrieve objects through a query sequence. These steps are:

1. Create a ConnectionManager
2. Create a connect to a specific JBI platform
3. Authenticate user credentials
4. Connect to the platform
5. Create a query sequence for a specific object type
6. Define query sequence parameters (including result set size for retrieved objects)
7. Activate the sequence
8. Define the predicate for the sequence
9. Issue the query
10. Consume the result set(s)

Integrated into this lengthy sequence is the time it takes to reassemble the objects from the relational table format into their XML metadata format. This overhead resulted in a retrieval time of approximately 30 seconds for 500 objects (per IO type) or 60 milliseconds per object. This time was approximately the same for both the current and proposed methods because the same platform query sequence methods were used once an IO type was known, and the tables all contained an equal number of objects. Furthermore, the queries were scoped to retrieve the same number objects for every IO type containing one to ten fragments. Due to the many matching IO types and this query execution overhead, the total execution often took hours to complete.

On the other hand, the execution time to retrieve the reference to all IO types containing one to ten fragments ranged from approximately 100 milliseconds to 20 seconds. Consequently, the measured improvement of the fragment technique would be lost in the overhead of the object retrieval if the method went beyond retrieving only the IO types vector.

المنارة للاستشارات
www.manaraa.com

Therefore, The evaluation only measured the execution time cost to obtain a single reference vector of every IO type associated with a particular fragment predicate sequence or fragment array. This ensured that no platform overhead costs for actual object retrieval were counted for either method. This will result in the most useful evaluation of the actual time savings of using the fragment index.

To evaluate the cost of these searches within the same context, the client application was implemented to retrieve the same subset of IO types with only one function call, so that the execution time for both methods could be compared in similar contexts. For instance, to test the current system cost to retrieve all IO types with geospatial content, the current method search through every IO type was done with one function call with a single predicate statement. For the proposed method, the application retrieved the array of IO types from the fragment table with a single function call.

The tests were run on every possible fragment combination, so that there would be a large sample of execution times for each possible fragment combination. However, because some combinations had small sets or were unique (i.e., for 10 possible fragments there was only one combination of all 10), there was less data available from these tests. This shortfall was alleviated by running multiple iterations of the same fragment combinations, if there was insufficient data to form a conclusion about the smaller result sets.

Table 4.4 is a simplified prediction of the cost savings of the fragment method. The number of searches in the table represents the number of evaluations required to obtain every possible matching combination of IO Types (e.g., if there was only one predicate composed of one fragment, one composed of two fragments, one composed of three fragments, etc., for a total of 10 predicate combinations). This comparison of methods applies to both of the aforementioned current MSR searching techniques, because each of those requires an evaluation of every IO type for a search. This table illustrates the obvious predictable outcome of the application test by demonstrating

المنارة للاستشارات
www.manaraa.com

Table 4.4: Cost Comparison of Methods

	Proposed	Current
Cost(# searches)	10	10230
Savings	1000%	

the order of magnitude by which the proposed fragment method would outperform the current iterative method.

4.5 Summary

This chapter contains an examination of the implementation of the fragment-based information engineering methodology. The preprocessing required to perform an evaluation of this research was to install and configure a sample JBI platform. After that was accomplished, a new Metadata Schema Repository (MSR) was created with the proposed fragment storage modifications. Additionally, sample data was added to the MSR and objects were published to the Information Object Repository.

After an evaluation technique was developed for the quantifiable elements of this research, a sample JBI platform was installed, configured and populated and an application was constructed to measure the performance of the fragment solution. The results and analysis of these tests are in Chapter 5. In addition to the application performance evaluation, there is a discussion regarding the components of this research for which the improvements are not measurable. Chapter 6 contains research conclusions and recommendations for integration of the solution and future work.

المنارة للاستشارات
www.manaraa.com

V. Results and Evaluation

5.1 Introduction

This chapter contains a discussion of the test results and evaluation of the fragment-based information object type engineering methodology. In addition to the performance evaluation, there is an analysis of the components of this research for which the benefits are not measurable, but are qualitative.

To provide contextual understanding of the test application, the testing environment is discussed first. Evaluation approaches and retrieval options are also explained before providing test results and analysis.

5.2 Testing

Before discussing the results of the performance evaluation program, the environmental factors and evaluation choices must be considered. This section provides an overview of the reasoning and the approach to the test development and some of the decisions and compromises that were made.

5.2.1 Testing Environment. The first decision that was made was to install and configure the platform on the same machine on which the testing would be performed. The reason for this was simply to allow ease of configuration and testing. Since no web services or traffic patterns would be evaluated, there was no reason to isolate the server from the client. This would ensure the execution times would only be limited by the resources of the machine and not network traffic patterns.

The Java programming language was used to build the test application. The JBI Core Services Reference Implementation Version 1.2 and the test program were executed on an IBM®-compatible machine with the following specifications and configuration:

- 1.6GHz Intel® Pentium® Processor
- 1.0 GB RAM

www.manaraa.com

- Windows® XP Professional
- MySQL Relational Database System Version 4.0.21
- Java Development Kit (JDK) 1.4.2
- JBoss Application Server 3.2.3 (part of JBI deployment architecture)

5.2.2 Evaluation Approach and Assumptions. One of the goals of this research was to allow for a multiple object type search using fragments. However, there was no method of comparison of this technique to the platform services. The current JBI architecture provides no mechanism for searching the metadata paths within IO types to find types which may fulfill the information needs of a JBI user. This would not preclude a platform user with proper privileges from developing a simple application to perform a linear search of IO types. However, for this to be a useful tool, the user would at least require some knowledge of metadata standards within the platform (e.g., geospatial latitude is represented as *lat* within this platform). For testing, it was assumed this was the type of user who would be using the current implementation.

It was also assumed that there were too many IO types to know which IO types contained any given metadata (other than some naming conventions and relative XML paths). This may be assuming more or less of what the typical user knowledge would be, but given the lack of any empirical data, this would be sufficient for a comparison of the current and proposed architectures.

5.2.3 Retrieval Options. As discussed in Section 4.4.7, the evaluation application was initially configured to retrieve a large set of information objects for each matching IO type, instead of just the IO type references. Since the object retrieval followed the same method for both the current and proposed architectures and was much more costly than the type retrieval, the application was modified to just retrieve the IO type references and insert each reference into a vector.

المنارة للاستشارات
www.manaraa.com

The choice of a matching IO type search method for the current system required an iteration through every type in the Metadata Schema Repository (MSR). As discussed in Section 4.4.2, there was a complication with regard to the validation of an XML schema against an XML instance. The other consideration (discussed in Section 4.4.3) was whether to use a single SQL or multiple SQL database query for this iterative search. Due to the absence of a method for IO type searching in the current platform architecture and interface methods, the current MSR evaluation was configured to provide the fastest possible performance using an iterative searching technique. Thus, the validation issue was resolved by inserting a blank instance of every schema into the MSR. This would provide the fastest method of evaluation, with only a minor MSR modification. The SQL query issue was resolved by comparing both methods against the proposed fragment solution. Using the blank instance and two SQL techniques demonstrates that the fragment methodology provides an improvement that could not be realized solely through the introduction of an iterative metadata searching technique.

5.2.4 Limitations and Validity. The test application was developed to evaluate this proposal "outside" the JBI platform. This means that the program would communicate directly with the MySQL database without the core web services or Common API (CAPI) methods. As discussed in Section 4.4.7, the platform requires quite a few steps to create a query, which is lengthier than the list of tasks required by the test evaluation methods. Furthermore, the testing conditions are idealized in the respect that every IO type has the same naming convention of a constant prefix and a suffix composed of a sequence of numbers identifying the fragments which are contained in that IO type. This fixed naming constraint eliminated the need to query the database for the list of IO type names. These generalities do not detract from the evaluation because the testing applies the same enhancements and limitations to all methods.

المنارة للاستشارات
www.manaraa.com

The choice of a 10 fragment initial library was a good choice to illustrate the hypothesized exponential savings of the fragment method, even though this is a much smaller library than may normally occur in practice. On the other hand, the absence of deployment statistics of the JBI RI limits the ability to speculate whether the 1023 IO type schemas tested here is large enough to explore the plausibility of this implementation and focus on whether the proposal addresses platform scalability.

The exhaustive use of every fragment combination was only a test parameter, and the ratio of the number of fragments in a platform library to the number of IO types in the platform MSR is likely to be much larger because some fragments will be used in very few schemas while others will be used in many. Thus, in an actual implementation, there will likely be many more fragments and the number of distinct IO type schemas tested here does not approximate the expected fragment distributions.

These questions illustrate the greatest challenge in this research endeavor–to evaluate and improve a system that is still under development. Compounding the lack of deployment statistics is the fact that some of the technologies used are still in their relative infancy (i.e., XML Schema Evolution and Web Services). The test application was built and scoped to provide the most accurate evaluation in spite of these limitations. For the areas which could not be quantitatively evaluated, this analysis includes substantial explanations for the hypothesized improvements.

5.2.5 Testing Procedure. Given an array of fragment names or a predicate expression composed of a random selection of each fragment's metadata, the test application was run for the three different MSR scenarios:

1. Query the 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 MSR to retrieve a large result set containing a schema instance, IO type name and version number for every IO type.

المنارة للاستشارات
www.manaraa.com

Iterate through the result set, testing each IO type in turn using the XPath validator method to see which types have matches for every metadata path in the predicate expression. Populate a vector with all of the matching IO types.

3. Iterate through the current MSR configuration with one SQL query to the MSR per IO type to retrieve a schema instance for that type. After each query, test the retrieved IO type instance using the XPath validator method to see if it matches every metadata path in the predicate expression. If there is a match, add the IO type to a vector of all matching IO types.

To allow for the two different techniques for evaluating the current MSR, the calling function was modified after testing the first method to evaluate the second method. As explained in Chapter 4, every fragment combination was tested. Due to the varying number of schemas containing a given number of fragments, two iterations were performed for each combination of fragments. This provided additional sample data when there was a smaller number of combinations. Table 5.1 shows the number of samples for each possible number of fragments for two iterations. The near symmetry of this table is due to the exhaustive use of every possible fragment combination and thus, the calculations shown. Where the execution time varied by more than 10% for the smallest sample size (2), the tests were re-accomplished for both samples to achieve a higher level of confidence for the execution time.

5.2.6 Test Results. Preliminary SQL tests were done in the MySQL Control Center database environment. These tests isolate the cost of the database interactions from the Java program overhead. For the proposed fragment method the SQL statement was the "SELECT..." statement developed for the "Get an array of IO Types for a given set of fragments" CAPI method discussed in Section 3.5.12, which contained the selection statement and a search predicate (the "WHERE" clause). For the current MSR multiple SQL query, a single full iteration of SQL statments was used to select each of the blank schema instances with the "known" IO type names (*known* because of the IO type naming conventions used in this research). For the

المنارة للاستشارات
www.manaraa.com

Table 5.1: Number of Samples per Number of Fragments

Number of Fragments in Array/Predicate	Number of Samples Calculation	Number of Samples Tested
1	2 * (10)C(1)	20
2	2 * (10)C(2)	90
3	2 * (10)C(3)	240
4	2 * (10)C(4)	420
5	2 * (10)C(5)	504
6	2 * (10)C(6)	420
7	2 * (10)C(7)	240
8	2 * (10)C(8)	90
9	2 * (10)C(9)	20
10	2 * (10)C(10)	2

current MSR single SQL query, a sample of 75 "SELECT *..." statement queries was executed which retrieved the schema instances, IO type names and versions of every IO type in the MSR. The average of these execution times is shown in Table 5.2.

In this table, the values are the same for both of the current MSR tests because the SQL statement did not depend on the number of fragments in the query. Since the multiple SQL query only varied by the IO type name in the search predicate and the execution times showed very little variation, only one iteration through all 1023 IO types was done for this method (each individual query took either 10 or 20 milliseconds). For the single SQL query, the execution time ranged between 160 and 200 milliseconds, per query, for the 75 identical test statements executed. The cost for these queries was much greater than any of the individual times for the multiple SQL queries because of the larger result set retrieved and the selection of multiple columns in the SQL query. There was a small enough variation between the 75 queries to accept the average execution time of 180 milliseconds. The proposed fragment MSR was the only case where variations in execution time must be considered. This is because each time a fragment is added to the search, the search predicate contains an additional test expression.

The results for the three scenarios tested in the Java program are shown in Figure 5.1. The longest execution times were for the multiple SQL query configuration.

المنارة للاستشارات
www.manaraa.com

Table 5.2: SQL Execution Times (sec)

Number of Fragments in Array/Predicate	Proposed MSR AVG SQL Time	Current MSR AVG Multi SQL Time	Current MSR AVG Single SQL Time
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

This was expected due to the base SQL query time for this method shown in Table 5.2. Much improvement was realized by the single SQL query method, although approximately 90% of the execution time was due to the iteration through the result set in the program. The proposed fragment method had the lowest execution time and most of that time was attributable to the SQL query. The program method cost only accounted for an average of 12% of the overall execution time. The fact that the fragment execution time increases with the number of fragments in the query is attributable only to the corresponding increase in the number of components in the search predicate of the SQL query. This factor will be discussed in greater detail in Section 5.3.

Using the slowest performance configuration as a baseline, Table 5.3 shows the improvement of the fragment method over both of the current MSR options. As hypothesized, the fragment solution provides improvement over the iterative search methods by at least a factor of 8. The execution time increases 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 that is the case, the *minimum* improvement can be stated.

المنارة للاستشارات
www.manaraa.com

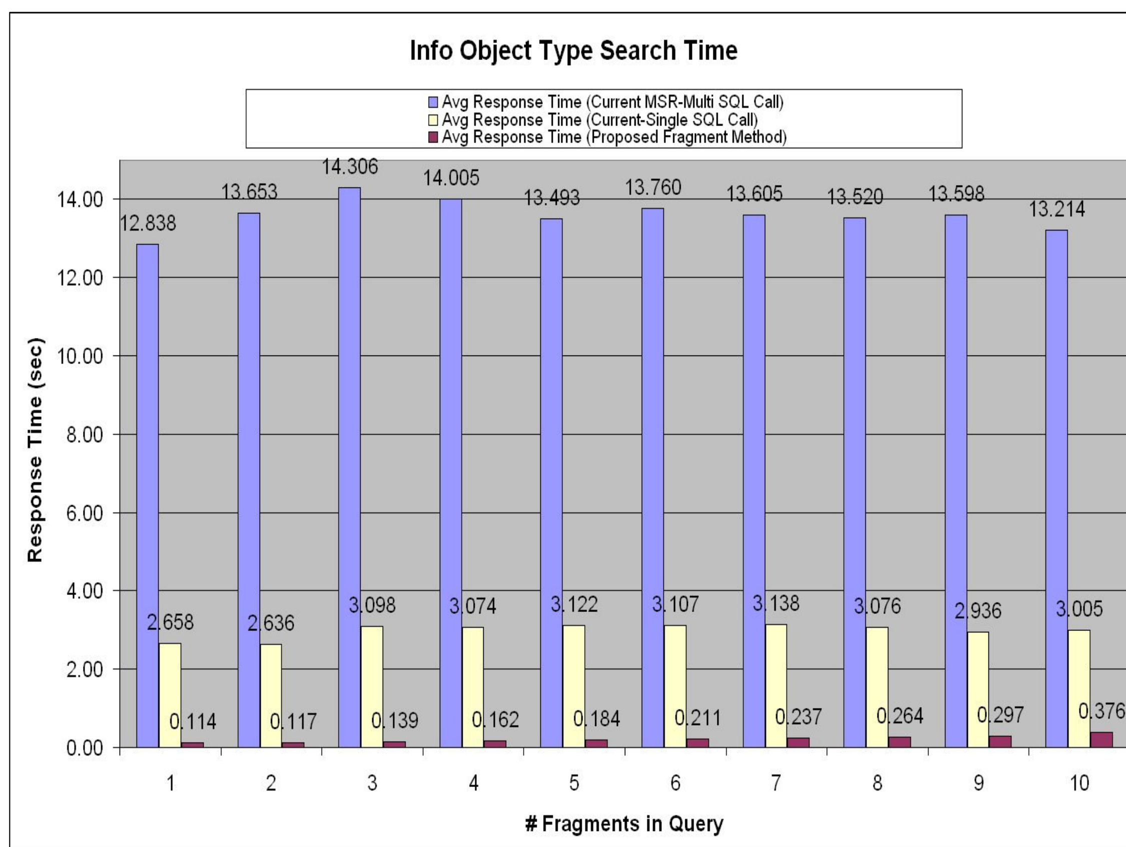


Figure 5.1: Info Object Type Search Time Comparison

5.3 Observations and Analysis

For the analysis of the test results, it is important to consider the actual measured improvement, as well any possible questions raised by the results and a cost/benefit analysis for implementation. In addition, the non-quantifiable elements of the proposal must be considered in any speculation about the comprehensive performance enhancements.

The test application used in this research required the introduction of a search method to the current MSR to demonstrate the improvement of the fragment method. Initially, this might be considered to be a cursory exercise because it is not a stretch to theorize that almost any technique would be an improvement over a linear search

المنارة للاستشارات
www.manaraa.com

Table 5.3: 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

method. However, the methodology must be considered in a broader sense by an additional evaluation of the peripheral benefits of the suggested implementations.

5.3.1 Search Time Improvement. 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.

The lengthier execution time is not due to an increase in the database size, which was constant for all tests. The number of expressions in the search predicate and the aggregation required by the "GROUP BY" function are the factors which 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 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 and what is the cost to execute a search with this number of fragments

المنارة للاستشارات
www.manaraa.com

in the predicate. This number should represent the maximum number of fragments contained in any particular IO type schema in a JBI platform 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 speculation, another test was conducted to expand the query results and allow a more confident 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 frament_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 100 queries with a 20 argument search predicate executed within the MySQL Control Center database environment. The average execution time for queries was 300 milliseconds. Comparing this to the execution time for 10 fragments in Table 5.2, 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

المنارة للاستشارات
www.manaraa.com

IO type schemas containing a total of at least 10,000 fragments 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.

5.3.2 Schema Revision Efficiency. The use of fragments allows the use of "includes" and "imports" in IO type schema development. This preserves referential integrity by allowing the use of fragment types and versions to track element and attribute changes. IO types only need to have version number changes in their new version files (noting the fact that schema file names must be unique within a namespace and included fragment file names will also have changed). The fact that there may be many IO types with a given fragment makes this much more efficient than making many changes to multiple schemas.

5.3.3 Schema Storage Improvements. While there are no proposed changes to the table field name in the MSR where the full IO type schema is stored, the use of includes and imports will make the IO type definitions only a fraction of the size of a fully expanded schema. For example, the base object fragment, which is part of every IO type, has 17 elements (many of which have element annotations), and is approximately 75 lines. The inclusion of this complex type as a fragment in a schema requires only 2 lines–one for the include or import declaration, and one for the complex type element declaration. A schema composed of fragments is also easier to immediately visually scan and identify the main components, because even with many fragments, it may still be viewable on a single page. Conversely, the base object alone spans almost 3 pages. The database implications of the reduced schema storage size requirements are that many more rows can be loaded into cache on a query because the row size will be smaller.

للاستشارات المنارة
www.manaraa.com

5.3.4 Reduced IO Type Knowledge Requirements. One of the best advantages to the fragment methodology is that it provides an additional mechanism to relate IO types apart from the package tree structure. The fragment_io table allows the introduction of multiple object type search techniques into the platform services. There will be no ambiguity with respect to inheritance because fragments will be level building blocks of IO type schemas. Rather than forcing IO types into a difficult to classify parent-child structure, fragments can be placed at parallel levels within an IO type schema and be equally identifiable and searchable.

This improvement also allows a user to identify fragments of interest without having to manually find IO types and identify their structure to build a query or subscription. The entire metadata path knowledge required to build an XPath predicate on all IO types with a particular fragment is contained within a fragment definition. All fragments will be defined at the same level within a schema (nested parallel to one another just one level inside the "metadata" element).

5.3.5 Versioning, Coercion and Namespaces. The benefits realized by the introduction of the standards and methods for versioning and coercion are not easily analyzed. First, neither technique is yet supported in the latest version of the JBI core services. However, both are in the planning stages for a later release. It was not difficult to envision the complexity in both areas that would be introduced by the addition of fragments. This research included basic standards and techniques for implementation to address the most complex issues. By formulating standards that allow IO type version changes to be directed by fragment version changes, this issue was greatly simplified. The table structure and example provided for coercion instructions also provides a basic framework for coercion implementation.

Full custom namespace support is not yet implemented into the JBI core services. Schema validation is supported as an option as deployed. The schema validation option provides for checking for schema well-formedness, ensuring the presence of compliant base object information, and making sure that a schema contains only

www.manaraa.com

supported simple types as elements. Instance validation is also supported and has various configuration options, including on or off, and how many IOs should be validated. Until namespace methods are implemented, it will not be feasible for platforms to properly support versioning because version changes will cause rejection of non-conforming schemas which may just be older, newer or platform specific versions of an IO. There are mechanisms to ensure unique IO type names within a platform, but full namespace support will allow a central namespace to contain a basic fragment library and prevent IO type and fragment name collisions. An additional benefit to the support of multiple and central namespaces is to allow multiple platforms to use similar or identical descriptive names for fragments and IO types. The uniqueness of a namespace ensures there will be no collisions in these situations. The fragment proposal also requires platform support for complex types in all namespaces, as this is the defined data type for all fragments.

5.3.6 Integration Issues. The integration of the fragment methodology requires changes to the query and subscription process. The positive aspect of this challenge is that the current method of querying and subscribing by IO type can remain unchanged. Whereas the current process requires an IO type and optional XPath predicate, the fragment change would require a fragment array and the optional search predicate. The flowchart for the new process is shown in Figure 5.2. An optional additional user input would be an array of acceptable payload types (e.g., string, image, etc.). This is not an explicit platform requirement, as the burden of acceptance or refusal of information objects (IOs) by payload type can be placed on the client side. This is not absolutely burdensome on the client, as there are potential benefits to this option, such as no requirement to modify subscriptions when requirements or processing capabilities change (only their IO processing applications would need to change). In any case, there are few new requirements for users and the platform integration requires that the current process flow be prepended with fragment processing methods. An important consideration for the payload format

المنارة للاستشارات
www.manaraa.com

issue is that some disadvantaged nodes (discussed in Section 2.2.1.4) may not have the ability to distinguish between payload formats to refuse or ignore the formats which they cannot process. In this case, a Guardian Agent should perform this filtering [24, 25].

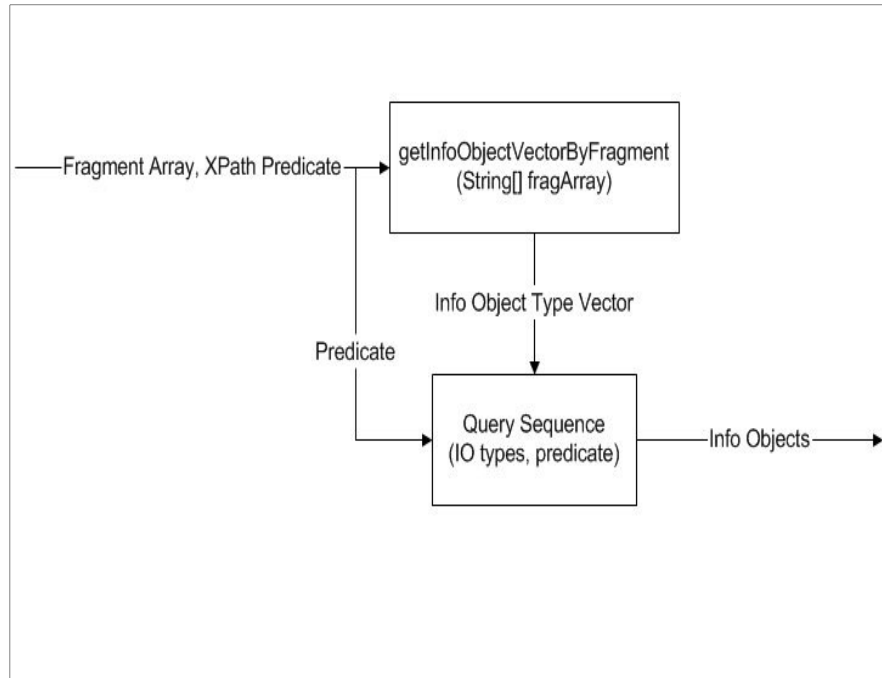


Figure 5.2: Fragment Query Process

If the suggested coercion techniques are implemented they will require modifications to publishing and subscribing processes. Since the proposed coercion rules allow publishers with active sequences to continue publishing old versions of IO types, the platform must use the coercion rules to convert the older IO versions to allow storage in the new table. This will require either an addition or modification to the publishing sequence. The same situation applies to active subscriptions except clients must be delivered their subscribed-to versions of IOs. Queries will only be allowed on new versions since they are only active during a current session and do not have a status maintained by the platform.

www.manaraa.com

The most expeditious way of implementing the coercion rules would be to create an appropriate sequence for all active subscribers and publishers at the time the new version is activated. That will allow potentially faster processing than if the coercion is done "on-the-fly". Since an update and publishing and subscribing may be happening at the same time, it is recommended that the old version tables continue to be used to store and retrieve IOs until all coercion tasks have been accomplished and conversion sequences have been created.

5.3.7 Shortfalls and Compromises. Some compromises are required to achieve improvements in the quality of services the JBI provides with these enhancements. For instance, additional storage will be required for the fragment tables. A user may have to provide some additional information for an object subscription or query (i.e., payload format), if subscription or query processing is modified to retrieve more than one IO type through a fragment query. Users should also be given the option of whether to retrieve objects of all matching IO types or only those that have payloads which they can process. This additional preprocessing may also be accompanied by a lengthier retrieval process for returning only the subscribed-to versions of objects to the user. However, this can be alleviated by allowing a payload format filter on all subscriptions.

A user may have to spend additional time to search for relevant fragments (versus relevant IO types) in other platforms if some cross platform search mechanism is developed, but right now there is no such searching tool. Therefore, any increased time for a fragment search must be weighed against not even having IO type information from unknown platforms without such a tool.

Any of the trade-offs required for implementing fragments and providing cross platform searching will be worth the cost in a large-scale JBI deployment because the assumption must be made that the volume of stored information object types will significantly outgrow any single user's knowledge of all IO types of relevance to all topics.

المنارة للاستشارات
www.manaraa.com

5.4 Summary

This chapter contains the results and analysis of a comparison of the MSR searching execution times for the current and proposed fragment-based MSR. As there are some improvements from this research that could not be evaluated with quantifiable results, the benefits and complications of the fragment implementation are also discussed to provide a well-rounded examination of the totality of the improvement. Chapter 6 contains some recommendations for future work and the conclusion of this research.

المنارة للاستشارات
www.manaraa.com

VI. Conclusion and Future Work

6.1 Introduction

This chapter contains recommendations for future work and research conclusions. The future work section is broken down into recommendations for different JBI platform architecture areas. Following the recommendations is a summarization of this research effort.

6.2 Main Research Contributions

The main contributions of this research relate to the new Information Object (IO) Type engineering methodology. The main improvements are:

- Simpler, componentized IO type development using fragments of metadata schema complex types,
- Introduction of new MSR storage architecture to include fragment definitions and fragment-IO type pairings
- Multiple IO type searches by fragments common to multiple types using a database table index of fragment-IO type pairings,
- Defined versioning and coercion standards for fragment and IO type evolution, and
- Simpler, distributed schema design and evolution.

6.3 Future Work

Future work opportunities are broken down into four main areas. Each of these areas contains topics which have been addressed throughout this research discussion. The focus in this chapter is on recommendations for the direction of future efforts in these areas.

6.3.1 Common API Expansion. Several new Common API (CAPI) methods are discussed in Section 3.5.12. A *Fragment* interface is suggested that will allow

المنارة للاستشارات
www.manaraa.com

the additions of the methods, *addFragment*, *addFragmentIOType*, and *getInfoObjectByFragments*. The *getFragmentByInfoObject* method should be added to the *InfoObject* interface.

The current method of creating a *sequence* on an IO type allows the establishment of publication and subscription exchange criteria between a client and a JBI platform. The integration of fragments into this sequence can be accomplished by allowing the creation of sequences on fragments as well as IO types. The fragment sequences would require the additional step of assigning an array of IO types (rather than a single IO type) to an activated sequence.

6.3.2 Fragment Library. The DOD Metadata Registry (DODMDR) and the Common Mission Definition (CMD) were discussed in Section 2.4. While the DODMDR does not contain a library of schemas in a common form, their Standard Data Element database contains 60,000+ elements with corresponding data types that are categorized by 9,500+ possible fragment-like groupings. These groupings contain from 1 to 151 elements per group and could become the initial fragments in a deployed library, which could be added to the JBI core services deployment platform. The relational database format of the DODMDR could be exploited to simplify the creation of the XML fragment schemas through a software program, since the database table contains fields for the element name, data type and description.

The 88 Common Mission Definition "fragments" are already in the standard XML schema format and are also already defined as complex types (as is the form of fragment definitions). Therefore, the only effort required to build a library of these fragments would be to acquire these schema files from the working group, initialize the base fragment versions and insert these into a fragment database that could then deploy with the JBI core services.

6.3.3 Platform Recommendations. The implementation of the versioning and coercion methods and namespace support must include support for the complex

المنارة للاستشارات
www.manaraa.com

data type and the include and import XML schema standards. As support and integration for these services is still in development, it would be useful to expand these efforts to include the fragment methodology.

6.3.4 Additional Exploration. One issue that was not investigated in this research but that seemed quite worthy of exploration is the use of references or pointers to IO type and fragment schemas in the database storage fields (instead of the full textual representation of these schemas). There are many improvements that could be realized with this change if relational databases continue to be the deployed access method. For one thing, it may reduce the quite lengthy time it takes to load the schema tree to the JBI IO Type schema list interface (Figure 2.4). Since the IO types are categorized by their directory-like package structure, this format is ideally suited to a basic file storage structure that would allow for faster schema retrieval.

Another issue that must be considered is that there may be fragments defined within other fragments (if a fragment complex type contains other complex types within its schema). Some of the CMD schemas are structured this way. At the initial consideration of this issue, it did not appear to pose any challenges as only the outer fragment will be stored in the fragment_io table pairings and the inner fragment is still a searchable node within the outer fragment by a properly formatted XPath predicate. However, if a user wishes to search on the inner fragment with the *getInfoObjectByFragment* method, there will not be any matches on fragment-IO type pairs in the fragment_io table. If this is resolved by processing every complex type within an IO type schema as a fragment, then the suggested coercion table Element_Node field will have to be redefined (since all fragments will no longer be parallel within a schema).

Most schema validators support the use of includes and imports and homogeneous, heterogeneous and chameleon namespaces. For this research, the open source Xerces parser/validator was tested, although the XPathEvaluator class provided by the JBI in-house team was used for XPath predicate path validation. Not enough has

المنارة للاستشارات
www.manaraa.com

been investigated about the JBI platform instance validation techniques employed by the platform to recommend this validation tool as a replacement for the current method. However, the blank instance validation method used in the test application proved to be fast enough to warrant further testing and analysis against the current method.

Another benefit of the schema instance field would be to use this instance to search for metadata terms in the MSR. The instance is in a simpler format for parsing and testing for key words or to build an index of metadata to fragments and IO types. This would be very simple to program and build using the XPathEvaluator method and would help new users to quickly search the fragment library for key words that would indicate whether a fragment may be useful for building or searching for IO type schemas.

6.4 Summary of Research

The main objective of this research was to improve the quality of service delivered by a JBI by introducing a new information engineering framework for information object (IO) schemas and thereby improve IO retrieval within a JBI. The introduction of this methodology provides other improvements, such as a technique for searching across related objects, more effective use of database storage, and reduction in time to build schemas, subscribe to and query for objects. In addition to these improvements, techniques were introduced to incorporate proposed versioning standards and coercion techniques into the platform architecture. The combined effect of promoting common libraries of these fragments across multiple platforms (i.e., in a central common namespace) will allow the reduction in IO type knowledge by users to do a more thorough search of objects in the entire available JBI information space.

المنارة للاستشارات
www.manaraa.com

	METADATA ELEMENT	HASHED FIELD NAME	INSTANCE 1	INSTANCE 2	INSTANCE 3	INSTANCE 4	INSTANCE 5
0	geospatial_frag\highElevatio	ior1263621817	1000	1000	1000	1000	1000
	geospatial_frag\highLat	ior_159236357	100	150	200	250	300
	geospatial_frag\highLong	ior_641346400	40	60	80	100	120
	geospatial_frag\humidity	ior1592171029	5	5	5	5	5
	geospatial_frag\lowElevatior	ior_1214708277	15	15	15	15	15
	geospatial_frag\lowLat	ior1639392141	50	100	100	100	100
	geospatial_frag\lowLong	ior_718437810	20	40	60	80	100
	geospatial_frag\visibility	ior340848628	4	4	4	4	4
1	target_frag\bda	ior_1268599297	4	4	4	4	4
	target_frag\ID	ior_1010754693	250	260	270	280	290
	target_frag\priority	ior_267762684	2	2	2	2	2
	target_frag\restrictions	ior1818493063	1	1	1	1	1
	target_frag\surfaceArea	ior1463200890	10	10	10	10	10
	target_frag\threat	ior_904793990	101	101	101	101	101
	target_frag\type	ior_671315558	20	30	40	10	30
	target_frag\vulnerability	ior540329500	100	110	110	110	110
2	weapon_frag\accuracy	ior551060644	100	100	110	100	100
	weapon_frag\cep	ior_819079741	5	5	5	5	5
	weapon_frag\optimalTargetT	ior_1627436990	2	2	2	2	2
	weapon_frag\pok	ior_819066943	75	75	75	75	75
	weapon_frag\size	ior378812716	20	25	30	40	50
	weapon_frag\weight	ior_922878749	30	40	50	60	70
3	ato_frag\aocID	ior1407383239	25	26	27	28	29
	ato_frag\dateExecuted	ior1405512838	null	null	null	null	null
	ato_frag\dateProduced	ior_1641266053	2004-12-10T00:00:00	2004-12-15T00:00:00	2004-12-20T00:00:00	2004-12-25T00:00:00	2004-12-30T00:00:00
4	aircraft_frag\ID	ior_1207153728	444	445	446	447	448
	aircraft_frag\payloadType	ior774919395	A2A	A2A	Supply	AntiTank	A2G
	aircraft_frag\range	ior_516704136	long	long	long	long	long
	aircraft_frag\restrictions	ior_1038655860	none	none	none	none	none
	aircraft_frag\type	ior_432227169	F15E	F16	C17	A10	C141
5	personnel_frag\afsc	ior1396295842	14S	62E	81S	23S	45E
	personnel_frag\assigned	ior_902384965	Y	Y	Y	Y	Y
	personnel_frag\name	ior1396678136	John Smith	Jane Doe	Sam Jones	Tom Taylor	Sally Simpson
	personnel_frag\rank	ior1396797337	Lt	Capt	Maj	SSgt	MSgt
	personnel_frag\unit	ior1396899057	8BW	8BW	8BW	8BW	8BW
6	bldg_frag\address	ior_85736936	444 Airway Rd	460 Airway Rd	522 Airway Rd	777 Airway Rd	566 Airway Rd
	bldg_frag\city	ior2018297287	Wright-Patterson AFB	Wright-Patterson AFB	Wright-Patterson AFB	Wright-Patterson AFB	Vright-Patterson AFB
	bldg_frag\installation	ior_13144426	null	null	null	null	null
	bldg_frag\name	ior2018617063	Hangar1	Hangar2	Hangar3	Hangar4	Hangar5
	bldg_frag\number	ior_1415749467	210	220	230	240	250
	bldg_frag\state	ior_1842207819	OH	OH	OH	OH	OH
	bldg_frag\zip	ior2004791109	45433	45433	45433	45433	45433
7	sensor_frag\ID	ior_633944924	888	822	722	755	733
	sensor_frag\platform	ior_974022340	GH	GH	Predator	Predator	Predator
	sensor_frag\reportIntervalSe	ior743766013	20	20	20	20	20
	sensor_frag\type	ior665619587	Motion	Motion	Heat	Heat	Heat
8	vehicle_frag\ID	ior640609428	333	343	353	363	373
	vehicle_frag\payloadType	ior_1177329009	null	null	null	null	null
	vehicle_frag\range	ior1894610724	1000km	1000km	1000km	1000km	1000km
	vehicle_frag\restrictions	ior_1428814240	none	none	none	none	none
	vehicle_frag\type	ior1446672499	Humvee	Humvee	Tank	Truck	Bus
9	report_frag\date	ior_1868505615	2004-12-10T00:00:00	2004-12-15T00:00:00	2004-12-20T00:00:00	2004-12-30T00:00:00	2005-01-05T00:00:00
	report_frag\expiration	ior1763588786	null	null	null	null	null
	report_frag\type	ior_1868006019	contact	contact	contact	contact	contact

Figure A.1: Fragment Data

www.manaraa.com

Appendix B. Source Code

This appendix contains selected java programs and SQL statements used in this research.

B.1 Fragment Create Table Statements

Listing B.1: fragmenttables.sql(appendix2/fragmenttables.sql)

```
# FRAGMENT CREATE TABLE STATEMENT

CREATE TABLE `fragment` (
  `fragment_schema` mediumtext NOT NULL,
  `fragment_name` varchar(100) NOT NULL default '',
  `ID` int(11) NOT NULL auto_increment,
  `fragment_version` varchar(100) NOT NULL default '0',
  PRIMARY KEY  (`ID`));

# FRAGMENT_IO CREATE TABLE STATEMENT

CREATE TABLE `fragment_io` (
  `ID` int(11) NOT NULL default '0',
  `fragment_name_version` varchar(200) NOT NULL default '',
  `io_type` varchar(200) NOT NULL default '',
  `io_type_version` varchar(200) NOT NULL default '',
 PRIMARY KEY  (`ID`,`io_type`,`io_type_version`),
 FOREIGN KEY (`ID`) REFERENCES `fragment` (`ID`));
```

B.2 Combination Generator

Listing B.2: CombinationGenerator.java(appendix2/CombinationGenerator.java)

```
/**
 * @author Michael Gilleland
 * Merriam Park Software
 */
import java.math.BigInteger;

public class CombinationGenerator {

        private int[] a;
        private int n;
        private int r;
        private BigInteger numLeft;
        private BigInteger total;

        //------------
        // Constructor
        //------------
```


المنارة للاستشارات
www.manaraa.com

```
public CombinationGenerator (int n, int r) {
        if (r > n) {
                throw new IllegalArgumentException ();
        }
        if (n < 1) {
                throw new IllegalArgumentException ();
        }
        this.n = n;
        this.r = r;
        a = new int[r];
        BigInteger nFact = getFactorial (n);
        BigInteger rFact = getFactorial (r);
        BigInteger nminusrFact = getFactorial (n - r);
        total = nFact.divide (rFact.multiply (nminusrFact)...
           );
        reset ();
}

//------
// Reset
//------

public void reset () {
        for (int i = 0; i < a.length; i++) {
                a[i] = i;
        }
        numLeft = new BigInteger (total.toString ());
}

//------------------------------------------------
// Return number of combinations not yet generated
//------------------------------------------------

public BigInteger getNumLeft () {
        return numLeft;
}

//-----------------------------
// Are there more combinations?
//-----------------------------

public boolean hasMore () {
        return numLeft.compareTo (BigInteger.ZERO) == 1;
}

//------------------------------------
// Return total number of combinations
//------------------------------------

public BigInteger getTotal () {
        return total;
}
```


للاستشارات
www.manaraa.com

```
//------------------
// Compute factorial
//------------------

private static BigInteger getFactorial (int n) {
        BigInteger fact = BigInteger.ONE;
        for (int i = n; i > 1; i--) {
                fact = fact.multiply (new BigInteger (...
                   Integer.toString (i)));
        }
        return fact;
}

//------------------------------------------------------
// Generate next combination (algorithm from Rosen p. 286)
//------------------------------------------------------

public int[] getNext () {

        if (numLeft.equals (total)) {
                numLeft = numLeft.subtract (BigInteger.ONE...
                   );
                return a;
        }

        int i = r - 1;
        while (a[i] == n - r + i) {
                i--;
        }
        a[i] = a[i] + 1;
        for (int j = i + 1; j < r; j++) {
                a[j] = a[i] + j - i;
        }

        numLeft = numLeft.subtract (BigInteger.ONE);
        return a;

}
public static void main(String []args){
        String[] elements = {"0", "1", "2", "3", "4", "5",...
            "6", "7", "8", "9"};
        int count = 0;
        int[] indices;
        for(int j = 1; j <= elements.length; j++){
                CombinationGenerator x = new ...
                   CombinationGenerator (elements.length, ...
                   j);
                StringBuffer combination;
                while (x.hasMore ()) {
                        combination = new StringBuffer ();
                        indices = x.getNext ();
```


www.manaraa.com

```
                        for (int i = 0; i < indices.length...
                            ; i++) {
                                combination.append (...
                                    elements[indices[i]]);
                        }
                        System.out.println (combination....
                            toString ());
                        count++;
                }
                System.out.println ("\nCount of all ...
                    combinations is: " + count);
            }
        }
}
```

B.3 Current MSR IO Type Search 1

Listing B.3: CurrentMethod.java(appendix2/CurrentMethod.java)

```
public static Vector getInfoObjectVectorBySearch(String[] ...
    allIOTypes, String predicate){
        Vector infoObjectVector = new Vector(0);
        String thisType = "";
        boolean matched;
        String metadata = "";
        try {
                Statement stmt;
                ResultSet rs;
                Class.forName("com.mysql.jdbc.Driver");
                String url = "jdbc:mysql://localhost:3306/mysql";
                java.sql.Connection con = DriverManager....
                    getConnection(url,"ior", "ior");
                stmt = con.createStatement();
                for (int l = 0; l < allIOTypes.length; l++){
                        thisType = allIOTypes[l];
                        String ioQuery = "Select schema_instance ...
                            from frag.ior_repository " +
                                        "where ...
                                            information_object_type...
                                             = '" + thisType + "'";
                        writeSQLDataToFile(ioQuery + ";\n");
                        rs = stmt.executeQuery(ioQuery);
                        while(rs.next()){
                                String instance = rs.getString("...
                                    schema_instance");
                                matched = XPathEvaluator.evaluate(...
                                    predicate, instance);
                                if (matched){
                                        infoObjectVector....
                                            addElement(thisType);
                                }//end if
                        }//end while loop
```


المنارة للاستشارات
www.manaraa.com

```
                }//end for
                con.close();

        }//end try
        catch(Exception e){
                System.out.println("Exception on msr query! "+e....
                   getMessage()+"  "+
                                e.getLocalizedMessage());
                e.printStackTrace();
        }

        return infoObjectVector;
}
```

B.4 Current MSR IO Type Search 2

Listing B.4: CurrentMethod2.java(appendix2/CurrentMethod2.java)

```
public static Vector getInfoObjectVectorBySearch2(String predicate...
   ){
        Vector infoObjectVector = new Vector(0);
        String thisType = "";
        boolean matched;
        String metadata = "";
        try {
                Statement stmt;
                ResultSet rs;
                Class.forName("com.mysql.jdbc.Driver");
                String url = "jdbc:mysql://localhost:3306/mysql";
                java.sql.Connection con = DriverManager....
                   getConnection(url,"ior", "ior");
                stmt = con.createStatement();
                String ioQuery = "Select information_object_type, ...
                   version, schema_instance from frag....
                   ior_repository";
                writeSQLDataToFile(ioQuery + ";\n");
                rs = stmt.executeQuery(ioQuery);
                while(rs.next()){
                        String instance = rs.getString("...
                           schema_instance");
                        thisType = rs.getString("...
                           information_object_type");
                        matched = XPathEvaluator.evaluate(...
                           predicate, instance);
                        if (matched){
                                infoObjectVector.addElement(...
                                   thisType);
                        }//end if
                }//end while loop
                con.close();
        }//end try
        catch(Exception e){
```


المنارة للاستشارات
www.manaraa.com

```
                System.out.println("Exception on msr query! "+e....
                    getMessage()+"  "+
                            e.getLocalizedMessage());
                e.printStackTrace();
        }
        return infoObjectVector;
}
```

B.5 XPath Evaluator

Listing B.5: XPathEvaluator.java(appendix2/XPathEvaluator.java)

```
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.List;
import org.dom4j.Document;
import org.dom4j.DocumentException;
import org.dom4j.DocumentHelper;
import org.dom4j.Node;
import org.dom4j.xpath.DefaultXPath;

/*
 * @author AFRL JBI In-House Development Team
 */

public class XPathEvaluator
{
        public static boolean evaluate(String predicate, String ...
            metadata)
        {
                // load appropriate evaluator
                DefaultXPath evaluator = new DefaultXPath(...
                    predicate);
                Document doc = null;
                try
                {
                        doc = DocumentHelper.parseText(metadata);
                }
                catch(DocumentException de)
                {
                        System.out.println("Exception parsing ...
                            metadata in XPathEvaluator: "+
                                    de.getMessage());
                }
                catch(Exception e)
                {
                        System.out.println("Exception parsing ...
                            metadata in XPathEvaluator: "  + e....
                            getMessage());
                }
```


المنارة للاستشارات
www.manaraa.com

```
                Object eval = evaluator.evaluate(doc);
                if(eval instanceof List)
                {
                        if(((List)eval).isEmpty()) return false;
                        else return true;
                }
                else if(eval instanceof Node) return true;
                else if(eval instanceof Boolean) return ((Boolean)...
                   eval).booleanValue();
                else
                {
                        // well at this point we have been given ...
                           an xpath expression that we really
                        // dont know what to do with... so... log ...
                           it and the type returned by the ...
                           evaluator
                        // so that we can do postmortem.
                        System.out.println("Well at this point we ...
                           have been given an xpath expression ...
                           that"+
                                        "we really dont know what ...
                                           to do with: "+
                                        "\n\tXPath Predicate: "+...
                                           predicate+
                                        "\n\tType Returned by ...
                                           evaluator: "+eval....
                                           getClass().getName());
                        return false;
                }
        }

}
```

B.6 Proposed MSR IO Type Retrieval

Listing B.6: ProposedMethod.java(appendix2/ProposedMethod.java)

```
public static Vector getInfoObjectVectorByFragment(String[] ...
   fragmentArray){
        Vector infoObjectVector = new Vector(0);
        //beginning of query statement
        String fragQuery = "SELECT distinct io_type, ...
           io_type_version, count(*) from  frag.fragment_io where ...
           fragment_name = ";
        //OR'd fragment = clause
        for (int i=0; i < fragmentArray.length; i++){
                if (i>0) fragQuery = fragQuery + " OR ...
                   fragment_name = ";
                fragQuery = fragQuery + "'" + fragmentArray[i] + "...
                   '";
        }
```


المنارة للاستشارات
www.manaraa.com

```
        //limit results to io_types which contain ALL the ...
           fragments (using count)
        fragQuery = fragQuery + " group by io_type, ...
           io_type_version having count(*) = " + fragmentArray....
           length;
        writeSQLDataToFile(fragQuery  + ";\n");
        try
        {
                //db connection info
                Statement stmt;
                ResultSet rs;
                Class.forName("com.mysql.jdbc.Driver");
                String url = "jdbc:mysql://localhost:3306/mysql";
                java.sql.Connection con = DriverManager....
                   getConnection(url,"ior", "ior");
                stmt = con.createStatement();
                rs = stmt.executeQuery(fragQuery);
                while(rs.next()){
                        String ioType = rs.getString("io_type");
                        String ioTypeVersion = rs.getString("...
                           io_type_version");
                        String ioTypeAndVersion = ioType + "_" + ...
                           ioTypeVersion;
                        infoObjectVector.addElement(...
                           ioTypeAndVersion);
                        }//end while loop
                con.close();
        }
        catch(Exception e)
        {
                System.out.println("Exception on fragment query! ...
                   "+e.getMessage()+"  "+
                               e.getLocalizedMessage());
                e.printStackTrace();
        }
        return infoObjectVector;
}
```


المنارة للاستشارات
www.manaraa.com

Bibliography

1. *Information Management to Support the Warrior.* Technical Report SAB-TR-98-02, USAF Scientific Advisory Board, Dec 1998. URL `http://www.rl.af.mil/programs/jbi/documents/IMReport.pdf`. SAB-TR-98-02.

2. *Mercury Capability Guidelines.* Technical report, AFRL/IF, January 2003.

3. "DOD METADATA REGISTRY and CLEARINGHOUSE", March 2005. URL `http://diides.ncr.disa.mil/xmlreg/user/index.cfm`.

4. "IFSE Branch, Systems and Information Interoperability", May 2005. URL `http://www.if.afrl.af.mil/div/IFS/IFSE/`.

5. "Joint Battlespace Infosphere (JBI) Common Application Programming Interface (API) Suggested Format for Version 1.5 (Draft 0.3)", January 2005. URL `http://www.infospherics.org/api/`.

6. *XML Inclusions (XInclude) Version 1.0 W3C Recommendation 20 December 2004*, May 2005. URL `http://www.w3.org/TR/2004/REC-xinclude-20041220/`.

7. *XML Schema Part 0: Primer Second Edition W3C Recommendation 28 October 2004*, May 2005. URL `http://www.w3.org/TR/xmlschema-0/`.

8. "Zero, One or Many Namespaces?" xFront XML Schemas: Best Practices, Jan 2005. URL `http://www.xfront.com/ZeroOneOrManyNamespaces.pdf`.

9. AFRL/IFSE. *JBI Quick Start Guide JBI Core Services Reference Implementation Version 1.2*, October 2004.

10. Bray, Tim, Dave Hollander, and Andrew Layman. *Namespaces in XML.* W3C, Jan 1999. URL `http://www.w3.org/TR/REC-xml-names/`.

11. Cattell, R.G.G. *Object Data Management Revised Edition Object Oriented and Extended Relational Database Systems.* Addison Wesley, 1994.

12. Costello, Roger L. "XML Schemas - XML Technologies Course". XML-DEV List Group, 2002. URL `http://www.xfront.com/BestPracticesHomepage.html`.

13. Eckstein, Robert. *XML Pocket Reference.* O'Reilly & Associates, Inc., 1999.

14. Gilleland, Michael. "Combination Generator". Merriam Park Software, Jan 2005. URL `http://www.merriampark.com/comb.htm`.

15. Harold, Elliotte Rusty. and W. Scott Means. *XML In a Nutshell A Desktop Quick Reference.* O'Reilly & Associates, Inc., 2001.

16. Hirschfield, Stuart H. "Developing Clients for the Joint Battlespace Infosphere". URL `http://www.rl.af.mil/programs/jbi/docs.cfm`.

المنارة للاستشارات
www.manaraa.com

17. Knoth, Briana, Vuhuy Phan, and Jerry Warner. *Common Mission Definition Information Model*. Technical report, 2004. Revision Number: Working Draft v1.291d.

18. Kokar, M.M. *Fusion as an Operation on Formal Systems: A Formal Framework for Information Fusion (FIFF)*. Technical report, Proposal to Air Force Office of Scientific Research, 1997.

19. Linderman, Mark H. and Paul T. Webster. "The Joint Battlespace Infosphere". URL `http://www.afrlhorizons.com/Briefs/June01/IF0018.html`.

20. Luo, R.C. and M.G. Kay. *Data Fusion and Sensor Integration: State-of-the-Art 1990s in Data Fusion in Robotics and Machine Intelligence*. Academic Press, 1992.

21. Marceau, Carla. "The JBI Information Object Type Heirarchy", June 2004. ATC-NY.

22. Marsh, Jonathon and David Orchard. *XML Inclusions (XInclude)*. W3C, version 1.0 edition, Nov 2003. URL `http://www.w3.org/TR/2003/WD-include-20031110/`.

23. Rosen, Kenneth H. *Discrete Mathematics and Its Applications, 2nd edition*. McGraw-Hill, 1991.

24. Satterthwaite, David E., Charles P. Corman and Thomas S. Herm. "Real-time Information Extraction for Homeland Defense". *7th International Command and Control Research and Technology Symposium*. Naval Post Graduate School, June 2002.

25. Satterthwaite, T.W.; Corman D.E.; Herm T.S.; Martens E.J., C.P.; Blocher. "IEIST force template technology provides a key capability for connecting tactical platforms to the global information grid". *Digital Avionics Systems Conference, 2004*, volume 2, 11.B.2– 11.1–9. Oct 2004.

26. USAF Scientific Advisory Board. *Building the Joint Battlespace Infosphere*, volume 1: summary edition, Dec 2000. URL `http://www.rl.af.mil/programs/jbi/documents/JBIVolume1.pdf`. SAB-TR-99-02.

المنارة للاستشارات
www.manaraa.com

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704–0188

The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704–0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202–4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. **PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.**

1. REPORT DATE *(DD–MM–YYYY)*
13–06–2005

2. REPORT TYPE
Master's Thesis

3. DATES COVERED *(From — To)*
Sept 2003 — Jun 2005

4. TITLE AND SUBTITLE
A JBI Information Object Engineering Environment Utilizing Metadata Fragments for Refining Searches on Semantically-Related Object Types

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

6. AUTHOR(S)
Felicia N. Harlow, Capt, USAF

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Institute of Technology
Graduate School of Engineering and Management
2950 Hobson Way, Bldg 641
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/GCE/ENG/05-03

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
AFRL
Attn: Mr. Robert Hillman, DSN 587-4961
525 Brooks Rd
Rome, NY 13441
robert.hillman@rl.af.mil

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT
Approval for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
The Joint Battlespace Infosphere (JBI) architecture defines the Information Object (IO) as its basic unit of data. This research proposes an IO engineering methodology that will introduce componentized IO type development. This enhancement will improve the ability of JBI users to create and store IO type schemas, and query and subscribe to information objects, which may be semantically related by their inclusion of common metadata elements. Several parallel efforts are being explored to enable efficient storage and retrieval of IOs. Utilizing relational database access methods, applying a component-based IO type development concept, and exploiting XML inclusion mechanisms, this research improves the means by which a JBI 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. The combined proposed framework provides a better means by which a JBI can deliver the right information to the right users at the right time.

15. SUBJECT TERMS
JBI (Joint Battlespace Infosphere), XML Schema, Metadata, Namespaces, Information Retrieval, Information Object, Web Services, Object Versioning, Schema Evolution

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Michael L. Talbert, Lt Col, USAF
U	U	U	UU	122	19b. TELEPHONE NUMBER *(include area code)* (937) 255–3636, ext 4613

Standard Form 298 (Rev. 8–98)
Prescribed by ANSI Std. Z39.18

المنارة للاستشارات
www.manaraa.com