AFRL-IF-RS-TR-2004-220
Final Technical Report
July 2004

INFRASTRUCTURE OPERATIONS TOOL ACCESS (IOTA)

Northrop Grumman Defense Mission

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

STINFO FINAL REPORT

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2004-220 has been reviewed and is approved for publication

APPROVED: /s/

RICHARD J. LORETO
Project Engineer

FOR THE DIRECTOR: /s/

JOSEPH CAMERA, Chief
Information & Intelligence Exploitation Division
Information Directorate

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	JULY 2004	Final Mar 03 – Sep 03

4. TITLE AND SUBTITLE
INFRASTRUCTURE OPERATIONS TOOL ACCESS (IOTA)

5. FUNDING NUMBERS
C - F30602-01-D-0167, 0015
PE - 63260F
PR - 2183
TA - QB
WU - 15

6. AUTHOR(S)
Steve Barth and
James Muller

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Northrop Grumman Defense Mission
12005 Sunrise Valley Drive
Reston Virginia 20191-3404

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Air Force Research Laboratory/IFEB
32 Brooks Road
Rome New York 13441-4114

10. SPONSORING / MONITORING AGENCY REPORT NUMBER
AFRL-IF-RS-TR-2004-220

11. SUPPLEMENTARY NOTES

AFRL Project Engineer: Richard Loreto/IFEB/(315) 330-3793/ Richard.Loreto@rl.af.mil

12a. DISTRIBUTION / AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

12b. DISTRIBUTION CODE

13. ABSTRACT ***(Maximum 200 Words)***

The purpose of Infrastructure Operations Tools Access (IOTA) effort is to accomplish development of an advanced architecture for ISR information management. The architecture will be based on a software component framework for information dissemination. The component framework will be based on Web Services standards and include provisions for enterprise workflow management. The component framework will leverage commercial off-the-shelf (COTS) technology and government off-the-shelf (GOTS) applications to provide mechanisms for data discovery, dissemination and visualization that can be rapidly adapted to varying information requirements. In particular, Java 2 Enterprise Edition (J2EE), will be used to provide a platform-independent set of standards and tools for developing the component framework. Broadsword Gatekeeper, a GOTS application, will also be used to define middleware capabilities for data access and discovery.

14. SUBJECT TERMS
Middleware, Data Source Access, Infrastructure

15. NUMBER OF PAGES
11

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Table of Contents

Section **Page**

SECTION 1 INTRODUCTION

The purpose of Infrastructure Operations Tools Access (IOTA) effort is to accomplish development of an advanced architecture for ISR information management. The architecture will be based on a software component framework for information dissemination. The component framework will be based on Web Services standards and include provisions for enterprise workflow management.

The component framework will leverage commercial off-the-shelf (COTS) technology and government off-the-shelf (GOTS) applications to provide mechanisms for data discovery, dissemination and visualization that can be rapidly adapted to varying information requirements. In particular, Java 2 Enterprise Edition (J2EE), will be used to provide a platform-independent set of standards and tools for developing the component framework. Broadsword Gatekeeper, a GOTS application, will also be used to define middleware capabilities for data access and discovery.

The infrastructure's development and deployment is a foundation for enterprise services for information management. Specifically, to provide a data access framework and metadata to production tools such as the Joint Targeting Toolkit (JTT) version 3.0, to provide Isaiah Phase 2 enhancements: automated cross-boundary data source access and integrated intelligence, and to provide data source access infrastructure for JEFX 04 and DGS-X.

SECTION 2 OBJECTIVES

2.1 Information Discovery

Find out what information products are available

2.2 Information Dissemination

Generate information products in response to requests or subscriptions

2.3 Information Management

Provide interfaces for monitoring and controlling information flow, and provide tools for generating, monitoring, and automatically satisfying information requirements (workflow management)

SECTION 3 ABBREVIATION / SYMBOL LIST

Axis An Apache programming interface for building and utilizing SOAP-based Web Services and provides a convenient mechanism for serializing arbitrary Java data as XML and transforming it to other text formats.

EJB *Enterprise Java Beans* – J2EE network-based component framework. Provides for distributed components as Java Beans.

J2EE *Java 2 Enterprise Edition* **–** a set of standards for building Java based services.

JAX-RPC *Java API for XML – based RPC* – is a Java standard for expressing Java class data as XML. This is used to encode Java objects so they can be passed via a SOAP message.

JMS *Java Messaging System* – J2EE programming interface and architecture for applications using message passing as an interface mechanism.

JCA *Java Connection Architecture* – a J2EE architecture for providing Java access to a wide variety of data sources and their available metadata.

SOAP *Simple Object Access Protocol* – standardizes sending data in XML and is the basis of most Web Services architectures.

UDDI *Universal Description, Discovery and Integration* protocol – directory service standard that provides metadata describing available Web Services and the mechanisms to access them.

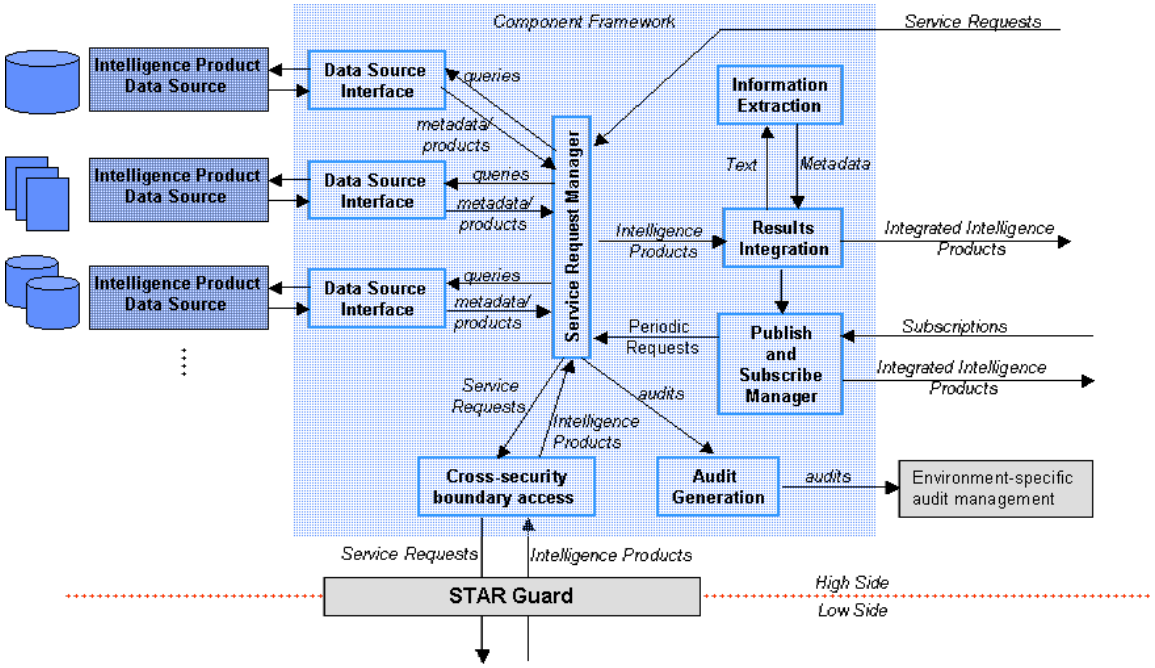
WSDL *Web Services Description Language* – a standardized XML Interface Description Language that describes the interaction with a data source in a message-based language-neutral format. This is often used in conjunction with SOAP in a Web Services architecture. Applications can use SOAP to obtain a WSDL description and then follow the description to obtain information from the Web service**.**

WSIF *Web Services Invocation Framework* – an Apache Java API for implementing WSDL interfaces. WSIF provides mechanisms that work with data sources that can be accessed via SOAP and also using EJBs, JCA, JMS, or Java classes.

XML *eXtensible Markup Language* – a standardized markup language for adding structured data to documents. XML is used to describe the information that a community of interest needs to exchange.

XSLT *eXtensible Style Language Transformations* - provide a standardized XML syntax for converting data from XML to different XML or non-XML text formats.

SECTION 4 IOTA SYSTEM ARCHITECTURE



4.1 Data Discovery Middleware

The IOTA effort has leveraged the Broadsword Gatekeeper capability to pull data to various display applications from multiple data sources through a common API. The components and component framework developed for this effort, have maintained the existing Gatekeeper API to support legacy Broadsword client applications, and provided for new API functionality using Web Services and XML documents to deliver information objects. New API functionality will include capabilities for publishing information to subscribing applications.

The infrastructure framework was developed in JAVA to be platform independent. Also, the use of AXIS and WSIF provides the foundation. They provide the mechanism to present access to multiple data source through a single view. Thus separating the user and applications from requiring a set of data source specific knowledge to perform their tasks. AXIS and WSIF are free and emerging as the standard web interface.

AXIS is a JAVA Web Services engine for processing SOAP messages. AXIS provides a server that can be run with most JAVA based application servers, such as Tomcat, WebSphere, Web Logic, Jboss, or EAServer. AXIS also has a flexible message processing chain and tools for generating the required service implementations from JAVA classes or a WSDL interface description and can use either HTTP or JMS for message transport. There is also a set of client classes to establish a connection to a service and process the requests and responses.

A SOAP message is received in AXIS by a handler class, whether the transport is HTTP, JMS, or something else. The handler determines what service the message is directed to and calls the correct provider class. The provider maps the method defined in the message to a JAVA method defined for the service, which is then invoked. The provider then serializes the return data into an XML SOAP response and sends it out through the output handler that, in turn, sends it to the requesting client. IOTA works by adding a custom provider class, which can process compound requests. The requests are converted into multiple back-end system calls, and the results returned as a single integrated response message.

The Web Services Invocation Framework (WSIF) is a framework for providing interaction with services through an abstract WSDL interface. Services can be simple JAVA classes, Enterprise JAVA Beans, JAVA Message Service (JMS) providers, or JAVA Connection Architecture components and WSIF can communicate with them using a common, message-based API. WSIF uses special bindings in the WSDL interface descriptions to connect and interact with the individual services. WSIF also has the capability of adding additional service types by extending the WSDL bindings and WSIF provider classes. IOTA utilizes WSIF in the IOTA Provider class in Axis to connect to the back-end data sources.

4.2 Data Source Access

The goal was to create a framework/infrastructure to provide access to any information source available. An information resource could be, but is not limited to: relational databases, messages, online documents, flat files, sensors, and imagery sources. Basically, any information that can be programmatically accessed could be made available through IOTA. Access to 3m, IPL, MIDB2.0, MDAL, and IET have all been successfully prototyped and demonstrated. One can successfully load a new information data source, by following the procedural steps to load a new data source.

4.2.1 Procedure to Load a New Data Source

1. Assumptions:
    a. Axis or some web service has been successfully deployed with a successful deployment of IOTA.
    b. The new data source has a JAVA API with public methods to access a data source with IOTAInputType and IOTAOutputType appropriately.
    c. IOTAInputType: Originally developed based on the ICML metadata standard. Can be extended to handle new required elements for a new data source. The IOTAInputType includes all relevant information required to retrieve the requesting data query.
    d. IOTAOutputType: Originally developed based on the ICML metadata standard. Can be extended to handle new required elements for a new data source. The IOTAOutputType includes all relevant information returnable to the requesting user or application.
2. JAVA API
    a. The API must use the IOTAInputTypes and IOTAOutputTypes in the class methods that will be used by IOTA to access the data source.

3. Generate WSDL for the data source from the JAVA API.
   a. Run the AXIS Java2WSDL application on the data source API classes to generate a WSDL (XML) document for the data source (DataSource.WSDL). Java2WSDL also generates files for a SOAP interface to access the API, which is used later on in the Test Client. The API should be defined in terms of the IOTAInputTypes and IOTAOutputTypes objects so that the requested data looks like the ICML standard. Using ICML required work-arounds for generating the WSDL document. (See 5.a.iii.2. below)
   b. Edit the DataSource.WSDL document. Normally the WSDL document provides the front-end definition for a Web service; however, while that's a desirable side effect of this process for testing and applications with single data source access requirements, the IOTA middleware needs to invoke the data source methods inside its own code.
   c. Modify specification for output types where substitutions were made to work around serialization problems. That is, wherever IOTAOutputTypes should have been used, but were avoided to allow Java2WSDL to work, they need to be put back in place of the XML tags and schema that were generated.
   d. For every SOAP binding, add a Java matching binding that will be used by IOTA middleware to invoke the data source service.

4. Generate the Web Services deployment descriptor for data source access.
   a. Run the WSDL2Java application on the DataSource.WSDL document completed in Step iii. This application generates the file used by AXIS to describe Web Services it handles. The descriptor file is an XML document named "deploy.wsdd".

5. Edit the deploy.wsdd
   a. Add a parameter naming the DataSource.WSDL file for the data source API (created in Step iii). The file name must exist in the directories on the JAVA CLASSPATH.
   b. Change the Service and Service Port tags to use the Java bindings instead of the SOAP bindings for the data source access methods. This involves pasting in the same code used in Step 5.a.iii.2.b.

6. Compile the JAVA API and put the class files into a jar file (DataSource.jar)

7. Edit the IOTA.wsdd file
   a. Provide access to the data source services for the IOTA middleware by integrating the data source deploy.wsdd document with existing IOTA.wsdd. Paste the Service tag definition from the data source deploy.wsdd document in to the IOTA.wsdd document. All information on the deployed service is contained in the structure of the Service tag. If one is not going to expose any methods of the new data source, skip to deployment of IOTA (4.2.1.9).

8. Expose the data source services directly for testing.

a. Add the SOAP bindings, from the DataSource.WSDL, for the data source services to the existing IOTA.wsdl. The IOTA.wsdl file defines the front-end IOTA services.
b. Update the AXIS config file by modifying the server_config.wsdd to add the SOAP operation tags generated in the DataSource.WSDL.
c. Update the code used by AXIS to run the service by editing IOTAEngineSOAPBindingImpl.java to add the method calls for the new interface and re-compile it.

9. Re-deployment
    a. Put the three files defining and implementing the service in the correct AXIS directories (see AXIS administrator for appropriate location): the IOTA.wsdd file, the DataSource.WSDL file generated in Step 2, and the jar file for the generated Java classes to include the IOTA.jar and the DataSource.jar.
    b. Stop and re-start AXIS. The IOTA.wsdd file is re-loaded and the new data source services are available for testing.
    c. Test the new data source services by using the Test Client Web page to define service by filling out a form and submitting it. When the form is submitted, the service is invoked and the XML document returned by AXIS is displayed as text on a Web page. The XML document displayed is the serialized objects returned from the data source service through AXIS. The XML document should include specification of the ICML tags for the data elements. A client application could then de-serialize the document to re-create the objects containing the data and accessible through methods corresponding to the ICML tags themselves.

4.2.2 Future Procedure

As one can see, the current procedure has numerous manual steps for editing key files along the way. It is our expectation to automate as many steps as possible.

4.2.3 Transformation of Data

Transformation includes the normalization of data and integrating the data for response.

4.2.3.1 Integrated Response

An integrated response is currently defined by concatenating data returned ICML normalized data from multiple data sources that were accessed to support a request.

4.2.3.1.1 Future Integrated Response

In the future an integrated response would be defined by collecting, merging, organizing, or processing data returned from multiple data sources that were accessed to support a request. The XML document returned as an IOTAOutputType would be based on the nomenclature of the requestor. In other words, the integration of the data would be based on the requestor preferences.

4.2.3.2 Normalization

Currently, the only normalization that occurs is within the JAVA APIs as data is returned from the data source. The APIs populates an IOTAOutputType that is currently based on the ICML format. This XML object is returned to the requestor as it is up to the requestor to parse the data appropriately.

4.2.3.2.1 Future Normalization

Each data source, legacy application, program, and/or operational working environment has its own nomenclature. For example, IPL queries on “Keyword”, the keyword in this case is a BE Number, MIDB uses the term “Facility”, which has a reference to a BE Number. Other applications use the term “Target ID” instead of a BE Number. However, in some targeting cells a Target ID is a facility and in other targeting cells a target it is a DMPI. It would be impossible given our current situation to get the whole DoD to use the same terms across the board. Therefore, it behooves us to map each element to a common set of terms that IOTA can use and understand to provide meaningful information to the requestor. In other words, normalize the requesting data, process the request and returned data, and then translate the response to a naming convention that the requestor understands. Therefore, it is incumbent on IOTA to understand the nomenclature of the requestors and data sources.

4.2.3.2.2 Normalization of the Request

Translate the elements of the request based on the nomenclature of the requestor. Pass the normalized request to the magic middleware where each appropriate data source would be queried for information.

4.2.3.2.3 Responding with the Requestor Nomenclature

Translate the elements of the response based on the requestor nomenclature. Thus returning a response that the requestor understands.

SECTION 5 DEPENDENCIES

Developing TTA functionality is dependent upon the ISSE Guard program. TTA requires interfaces to the ISSE Guard application. No other dependencies have been identified.

SECTION 6 SUMMARY AND CONCLUSION

This report describes the progress achieved on IOTA development under this task. Follow-on efforts are continuing to develop IOTA capabilities under the Intelligence Community Engineering (ICE) Task Order contract.