BUILDING FEDERATED BIOINFORMATICS DATABASES USING WEB SERVICES

by

CARY MARCUS PENNINGTON

(Under the Direction of John A. Miller)

ABSTRACT

Laboratories around the world continue to generate massive amounts of genomic and functional genomic data. Access to these data resources via Federated Databases through the Web is an important emerging technology. Federated databases allow several databases to be integrated while not discarding existing databases or losing local control over the administration of the databases. In order to achieve this desirable goal, technology needed to emerge to handle the heterogeneity and local autonomy of distributed database systems. Web service technology, including semantic Web service technology, provides a new opportunity to make federated databases a practical reality. This thesis presents architecture and an implementation to build federated databases using Web services. In particular, it demonstrates how Web service technology can provide the flexibility to create a dynamic federation of databases with sufficient abstraction to maintain the autonomy of the component systems and robustness to handle the heterogeneous nature of the disparate databases. A case study is conducted that involves federating six existing bioinformatics databases, CryptoDB, GiardiaDB, PlasmoDB, ToxoDB, TrichDB and TryTrypDB, to create the EuPathDB (formerly ApiDB) federated database.

INDEX WORDS: Web Services, Data Integration, Database, Federation, Bioinformatics

BUILDING FEDERATED BIOINFORMATICS DATABASES USING WEB SERVICES

by

CARY MARCUS PENNINGTON

B.S., Furman University, 1998

A Thesis Submitted to the Graduate Faculty of The University of Georgia in Partial Fulfillment

Of the Requirements for the Degree

MASTER OF SCIENCE

ATHENS, GEORGIA
2009

BUILDING FEDERATED BIOINFORMATICS DATABASES USING WEB SERVICES

by

CARY MARCUS PENNINGTON

Major Professor: John A. Miller

Committee: Eileen Kraemer
Jessica C. Kissinger

Electronic Version Approved:

Maureen Grasso
Dean of the Graduate School
The University of Georgia
December 2009

ACKNOWLEDGEMENTS

I would like to thank Dr. John Miller and Dr. Eileen Kraemer for their patience and help with this thesis. Also Cristina Aurrecochea, Steve Fischer, Xin Gao and Dr. Jessica C. Kissinger for their support of my work on the EuPathDB Project.

I also want to thank my family for their help and patience while I completed this milestone of my education.

TABLE OF CONTENTS

Page

ACKNOWLEDGEMENTS iv

LIST OF TABLES vii

LIST OF FIGURES viii

CHAPTERS

1 INTRODUCTION 1

2 BACKGROUND 6

2.1 Database Federation 6

2.2 Federated Databases in Bioinformatics 8

2.3 Web Services 11

2.4 Web Services in Bioinformatics 13

3 MOTIVATION 16

3.1 EuPathDB Project 16

3.2 History of EuPathDB and Problems 20

4 ARCHITECTURE 22

4.1 Web Services Federation 22

4.2 Case Study: EuPathDB Architecture 26

4.3 Incorporation of Outside Databases 33

4.4 Supporting Workflows 35

5 EVALUATION 38

6 CONCLUSIONS AND FUTURE WORK 42

BIBLIOGRAPHY 45

APPENDIX A: IMPLEMENTATION DETAILS OF EUPATHDB WEB SERVICES 49

APPENDIX B: API DOCUMENTATION 52

LIST OF TABLES

Page

Table 1: Approaches to Implementing Web Service Integration 23

Table 2: Maintenance Comparison For EuPathDB Project 32

LIST OF FIGURES

Page

Figure 1: Basic Layout of Federation System 7

Figure 2: Screen Shot of EuPathDB.org Home Page 16

Figure 3: Primary Key Query Architecture At a Component Site 18

Figure 4: PlasmoDB SQL Listing of GenesByMetabolicPathways 19

Figure 5: ToxoDB SQL Listing of GenesByMetabolicPathways 19

Figure 6: Architecture of EuPathDB Federation 28

Figure 7: Data Mediation 34

Figure 8: Search Strategy System on EuPathDB.org 37

Figure 9: Listing of invokeEx() Function 49

Figure 10: Listing of execute() Function 50

Chapter 1

Introduction

Data are a critical part of any software system and the storage of data is a topic that all system designers must consider. Today, data are being created at a rapid pace. Industries like medicine that formerly relied on paper charts for documentation and film for evidence are turning to electronic versions of these data. Genetics has seen a similar explosion in data generated through advancements in sequencing technology. As these massive amounts of data are gathered, they are stored in databases that are usually associated with the people that gathered that information. This is very useful to the curators of the information; however, it would be ideal for that information to be available to others to further their work as well. To gain access to all of this data, often researchers must access multiple databases, then collect and analyze the information to find the evidence they need to carry out their work. This can be an error prone and time-consuming process. The goal of federated database systems is to consolidate these data into a single, logical unit that can be accessed by an individual or computer system to obtain data. Note, this is a “logical” unit as opposed to a physical unit. Data are not being moved or copied to the federating system for purposes of storage; though, data could be duplicated in the name of performance or reliability. The primary advantage of federation is that the data are kept in their original location and continue to be curated by the personnel that know the data best.

A federation is the transparent integration of multiple data sources into a single logical resource. These resources can have different schema, different locations, and different semantics (Sheth and Larson, 1990). Each of these differences creates a new problem for the federation system to handle. These problems have been more formally described with the following terms: autonomy, distribution and heterogeneity. According to (Sheth and Larson, 1990), there are four types of autonomy.

- Design autonomy: Local databases can choose the data model, query language, attributes, etc. for their system.
- Communication autonomy: Local databases decide on their own when and how to communicate with other components of the federated database system.
- Execution autonomy: Execution of local operations/transactions will not be influenced by external operations.
- Association autonomy: Local databases can decide how much of their data/functions/operations to share with other components of a distributed database (Sheth and Larson, 1990).

Preserving these autonomies will ensure that the local application of the component databases is left undisturbed by the federation. Heterogeneity is another challenge for federated databases. Two types of heterogeneity standout: Structural and Semantic. Structural heterogeneity describes differences in schema, query language, transaction protocol, and database type (relational, object-oriented, or object-relational). The most accepted approach for solving schema heterogeneity is global view/schema integration; that is, mapping local schemata to a single global schema. However, creating a global schema that is accurate, flexible and concise is a difficult task made

more difficult by the existence of semantic heterogeneity. Semantic heterogeneity defines the idea that different meanings can be attached to the same term in different systems. Ontologies are an appropriate solution for handling this type of heterogeneity. Ontology is a form of knowledge representation that formally describes objects and the relations between them (Gruber, 1991). In the biological community, many ontologies already exist including the Gene Ontology (GO: http://www.geneontology.org/) which provides a controlled vocabulary to describe gene and gene product attributes in any organism, the Sequence Ontology (SO:http://www.sequenceontology.org/) which aims to develop an ontology suitable for describing biological sequences, and the Open Biological Ontologies (OBO: http://obo.sourceforge.net/#rel) which is an umbrella Web address for well-structured ontologies used across different biological domains.

Compared with other integration approaches, such as data warehousing, federated databases provide a flexible and robust solution to these problems (Wang et al., 2009). Furthermore, several implementations of federated databases were compared and it was found that Web services were competitive in performance while providing a very flexible solution (Wang et al., 2009).

For these reasons, we decided to use federated database strategies to accomplish our goals within the EuPathDB (eupathdb.org) bioinformatics project (Aurrecoechea et al., 2007). EuPathDB is a project funded under contract from the National Institutes of Health (NIH). It is charged with creating bioinformatic resources to aid scientists in their research of eukaryotic pathogens. The EuPathDB project currently consists of 6 database-driven Web sites, which represent 21 different eukaryotic parasites. These species include *Toxoplasma gondii* and *Plasmodium faliciparum* that

cause toxoplasmosis and malaria, respectively. Along with these sites, there is the EuPathDB.org site, which is a portal to access the component sites. This portal is the subject of a case study in this thesis. With this system in place, EuPathDB has expanded from servicing 3 sites to 6. This portal not only provides the communities with a single URL to access all the data within the EuPathDB projects, but also allows the researchers to mine data by asking questions across all of the sites to discover cross species relationships. Researchers worldwide use the EuPathDB resources to study the pathogens they represent in an effort to develop more effective therapeutics and advance basic research.

As one would expect, the above-mentioned problems of heterogeneity and autonomy presented themselves in this project. Since the component databases were already in active use, we needed a means to integrate them without interfering with their current functionality and management. Though all the component sites are based on a single schema, the Genomic Unified Schema (GUS), the differing ways in which the schema was applied and naming inconsistencies forced us to confront many of the challenges of federating databases. The use of Web services to implement the federation puts the responsibility regarding how a database represents its data in the hands of the people who know that data best, but restricts them to providing an interface with which the federation system can communicate. Local administrators are free to configure and use their systems as they see fit, with the only constraint being that they maintain a stable and abstract implementation of the interface provided by the federation. Web service federation aligns the incentives of all the parties involved so that a comprehensive database federation system is possible.

This thesis will describe a novel approach to implementing a Web service based federation system. Chapter 2 will provide some background information including other work that has been done in this field and how our project relates. In chapter 3 the motivation for the project will be discussed. Chapter 4 will describe the details of this approach to federated databases and explain the implementation that is being used on the EuPathDB.org site. Chapter 5 will evaluate the benefits and disadvantages of this system and chapter 6 will provide the conclusions that were drawn from this work.

Chapter 2

Background

2.1 Database Federations

Database federation was an idea coined by Hammer and McLeod in 1980 (Hammer and McLeod, 1980). A database federation is created through the transparent integration of multiple autonomous databases into a single, logical system (Sheth and Larson, 1990). These systems are composed of a federation system, or mediator, and component systems. Figure 1 shows the connections between these systems in a basic database federation. It is important to note that the links in the image represent live connections or the ability to make a live connection to the database and that the component systems are not identical type systems. Such systems should provide data abstraction to a point that the user of the federated system will view the data as being from a single source. Also, since the constituent databases are autonomous systems, the management system and query languages are not guaranteed to be the same (Sheth and Larson, 1990). Thus the system must decompose queries and translate them into the native query language of the component system. These issues lead us to the three major problems that must be solved by a federated database management system: Heterogeneity, both structural and semantic, Distribution and Autonomy (Sheth and Larson, 1990).

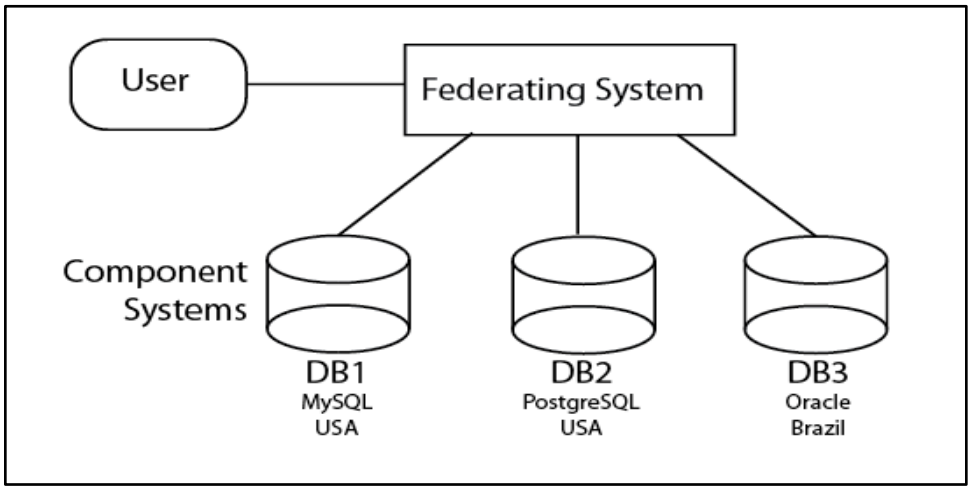

Figure 1: Basic Layout of Federation System

Heterogeneity is the concept that the data, though related, are not guaranteed to be stored in the same manner on all data sources. In one system, data could be in a weakly structured textual format, in a semi-structured document (like XML) or in a well-structured database management system. Even well structured databases can store equivalent data according to differing schemas. We will refer to this type of heterogeneity as structural heterogeneity. It is the difference in the manner in which the data is stored. Different media or differing table structures in a Database Management System (DBMS) pose the same problems. They both make it difficult for an outside system to interact with data without intimate knowledge of the system housing the data. Semantic heterogeneity is another type of heterogeneity that poses a challenge to federation. As noted above, semantic heterogeneity is the idea that identical terms could have very different meanings. Even in systems that are in the same domain, terms and definitions will not always match.

Distribution is another topic for a federation system to cover. The fact that the involved systems are not located in close proximity adds to the federation's complexity. This physical separation makes maintenance a challenge that is exacerbated by communication issues amongst the individuals responsible for that maintenance. As systems are altered or updated in the name of maximizing up time, the impact on a distributed integration project is often overlooked.

Autonomy is defined as the ability for an individual to make a rational, non-coerced decision. This applies to our database systems as well. The federating system knows less about the constituent data and thus should allow for local control over all administrative features of the local data.

2.2 Federated Databases in Bioinformatics

Despite the above challenges, federated systems are increasing in popularity for integrating bioinformatics information systems partly due to the success of some high visibility implementations (see below). Utilizing differing techniques, many organizations have implemented database federations to provide access to their data. A few examples of such systems are BioMart, Entrez, CytoScape, Galaxy and Neuroscience Information Framework (NIF).

Entrez is a cross-database federated search mechanism implemented at National Center for Biotechnology Information (NCBI). This system allows a global query interface covering over 35 databases containing over 350 million records (Sayers et al., 2009). Amongst theses databases are PubMed, PubMed Central, GenBank, and Protein Database. The initial interface to Entrez is the global query interface where any query is executed across all databases. Entrez extends the same simple query

interface to each of its component systems as well. A “Limit” feature is available to allow results to be narrowed to a more manageable set (Sayers et al., 2009).

NCBI has also provided a programmatically accessible version of Entrez with the Entrez Programming Utilities. These utilities are composed of eight applications deployed on the Web that accept parameters for querying the databases in Entrez. The “einfo” endpoint provides information about a given database. The “esearch” service accepts a text query and returns the results of the query as identifiers that can be fed into “efetch” or “esummary” to get detailed information about the records (Sayers et al., 2009).

BioMart is a query-oriented system used to integrate disparate types of data. BioMart was originally developed for Ensembl as EnsMart. EnsMart was developed as a data warehouse for EBI’s and Sanger’s internal data along with some outside data-sources. Since then it has developed into a generic data integration package known as BioMart. BioMart consists of a package that can be installed to provide access to data (Smedley et al., 2009). The same package can be used to provide access to local data or remote data. The BioMart Central application, described later, is an example of remote access through BioMart. Access is available through two main avenues: MartView and MartServices. MartView is the web interface to the BioMart installation. It allows a user to interactively build a query against any datasets that are available to that BioMart installation. MartView also allows the query to be downloaded by the user as XML to be used as input to MartServices. MartServices are RESTful and SOAP based Web services (Smedley et al., 2009). Both implementations support having the

MartView XML POSTed to the service URL and will return the same results as the MartView interface. This allows for distributed access to any BioMart installation.

BioMart consist of two main subsystems: QueryPlanner and Aggregator. The QueryPlanner is responsible for accepting a query and breaking it down for distribution to the appropriate component systems. If the planner detects credentials for a direct connection to the database, then it will formulate SQL for the query and retrieve the results. Otherwise, an XML request is built and the results are retrieved from the remote system via a Web service implemented through the BioMart system. The Aggregator receives the responses from the SQL and Web service queries and combines all of the results into a single result. All columns are mapped to 'exportables' and all filters are mapped to 'importables'. This abstraction hides the semantics of the output columns, and thus allows the results of the component systems to be easily combined (Smedley et al., 2009).

BioMart is mentioned here as an example of a distributed Federation system though it still maintains a vast data warehousing aspect as a result of its origins as EnsMart. BioMart provides an interface to allow standard access to the data within a component database. This type of access makes data publicly available. It is this public interface that has made BioMart such a popular choice.

Several other integration systems are using BioMart to integrate biological data in different ways. CytoScape is one such system. CytoScape has developed a visualization system that enables users to view relationships between entities graphically. CytoScape is applicable to many fields, but is used in conjunction with BioMart data sources to gather and display biological data.

Galaxy is another example of a system that utilizes BioMart systems amongst other data sources (Giardine et al., 2005). Galaxy is a portal system developed by Penn State researchers in conjunction with a group from Emory University. Galaxy provides a list of data sources that can be accessed from the Galaxy web interface. The caretakers of these data sources have made modifications according to the Galaxy API to link into the Galaxy system. A few examples of linked sites are UCSC, Flymine, and BioMart. Galaxy also supports tools used to analyze data collected from these disparate data sources (Giardine et al., 2005).

2.3 Web Services

A Web Service is defined by the World Wide Web Consortium (W3C) as "a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface that is described in a machine-processable format. Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using Hyper Text Transport Protocol (HTTP) with an XML serialization in conjunction with other Web-related standards" (Haas and Brown 2004). Simple Object Access Protocol (SOAP) is an Extensible Markup Language (XML) specification for message-based transactions over a network.

Web services are available in two implementations, RESTful and SOAP, both of which are widely accepted in both academia and industry. Representational State Transfer (REST) services are very lightweight services that allow for simple implementation. These services are confined to only the standard HTTP operations of GET, PUT, POST and DELETE. REST services work by binding methods within the service to the HTTP operations. This simplicity is accomplished by standards like JAX-

RS, which handles the binding of JAVA code to the standard HTTP operations. Jersey is a reference implementation of the JAX-RS standard (Hadley and Sandoz, 2009). For example, to implement a Java class as a REST service, using JAX-RS, the developer need only annotate the source code to indicate which methods are called on a given operation. If a user accesses the service via the URL with a GET request, the method that was annotated for GET is invoked. RESTful services require only a Web server to be functional. SOAP services are also known as “big services”. This architecture got this name from that fact that it requires more back-end infrastructure to support it. For the remainder of this thesis, the term “Web service” will refer to SOAP-style services but it could just as easily be applied to RESTful services. SOAP services utilize a stack of standard protocols, known as WS-* to describe the interface used to interact with the service, security, and constraints amongst other features.

The Web Service Description Language (WSDL) is a well-defined XML specification that describes the service in sufficient detail that the reader of the file could invoke it (Curbera et al., 2002). WSDL is the cornerstone and the container or attachment point for most of the WS-* protocols. WSDL has a new flavor that is a standard from W3C called Semantic Annotations for WSDL (SAWSDL). This standard is based largely on work performed as a joint venture between IBM and the University of Georgia. This work proposed a standard called WSDL-S which adds semantic annotations to WSDL files from ontologies using “modelreference” attributes on elements of the WSDL standard (Akkiraju, et al., 2005). Many features of WSDL-S were adopted in the SAWSDL standard. SAWSDL allows similar annotations to provide meaning to the terms in the WSDL file (Kopecky et al., 2007).

SOAP is the transport protocol for Web Services (Curbera et al., 2002). A SOAP message contains header information for the routing message and the payload, which is either the input or the output of the service as described in the WSDL file.

Universal Description, Discovery and Integration (UDDI) is a specification for a registry that allows for Web services to be published. From here, others can discover and download the WSDL for the published services. Information from the WSDL file is used to invoke the service. (Curbera et al., 2002)

Many other standards exist to define attributes of Web services. WS-Policy, WS-Addressing and WS-Transaction are only a few of the members of the WS-* protocol stack that are used to further define and advertise a service.

Since Web services are publishable entities, they can be discovered (Aggarwal et al., 2004). The METEOR-S project at the University of Georgia (Verma et al., 2005) was designed and implemented to perform just such a function. Radiant is a software package that offers a GUI for annotating WSDL files into WSDL-S or SAWSDL. METEOR-S utilizes keyword and semantic searching of annotated and non-annotated services to discover those that are compatible with the search criteria. Once the services are found, WS-Policy can be used to establish which of the discovered services are the "best" choices to be added to the federation. This discovery functionality was implemented in the Lumina package (Li et al., 2004).

2.4 Web Services in Bioinformatics

With the growth in biological data, more database systems are beginning to offer Web service based interfaces. Three such projects are described below. Each has implemented Web services in a different way and to a different extent. In addition to

these three examples, the National Center for Biotechnology Information (NCBI), European Bioinformatics Institute (EBI), and the DNA Data Bank of Japan (DDJB) have implemented Web service access to their data.

KEGG (Kyoto Encyclopedia of Genes and Genomics) is a bioinformatics site seeking to integrate 7 databases using a graph structure (Goto, 2000). These 7 databases are broken down into the Gene universe (comprised of the GENES, SSDB, and KO databases), Chemical Universe (COMPOUND, GLYCAN, and REACTION databases) and the Protein Network (PATHWAY database). KEGG provides an API, which is a SOAP interface to query the KEGG system (Goto, 2000).

PathPort, which is short for Pathogen Portal, is a system designed and built at the Virginia Bioinformatics Institute to provide analysis and visualization of infectious disease data. PathPort works along side ToolBus, a client-side application used to connect to PathPort. PathPort is implemented as a group of Web services, each of which provides the answer to a certain question or query. ToolBus provides a means of using these services, but they can be invoked from the WSDL file and executed without ToolBus as well (Yang et al., 2006).

BioMOBY is a system that allows the interoperability between biological data hosts (Wilkinson, et al., 2008). It works by using Grid and Web service technologies. BioMOBY has become popular and today many Web services and systems are able to interact using its set of standards. BioMOBY offers many tools to access its repository of services including a Java API to allow programmatic access (Wilkinson and Links, 2002).

The work mentioned above mostly represents Web services being used in the bioinformatics sector. This project's topic is a slight variation on that work in that our services are used internally to link systems together to create a single point of access to a vast amount of data. In the process of doing this, we have created a system that supports publicly accessible Web services that can be used by developers and researchers.

Chapter 3

Motivation

3.1 EuPathDB Project

Figure 2 Screen Shot of the EuPathDB.org Home Page

EuPathDB, formerly known as ApiDB (Aurrecoechea, et al., 2007), is a bioinformatics database system that provides genomic and functional genomic data on eukaryotic, parasitic pathogens. It is a portal site that allows access to data at six component sites. Each of these component sites has a domain, website, and users of its own. EuPathDB communicates with the component sites to retrieve data in response to queries that are asked from the EuPathDB interface. Currently, the system

consists of six individual databases, PlasmoDB.org which works with *Plasmodium* species (Aurrecoechea et al., 2009), CryptoDB.org, for *Cryptosporidium* species (Heiges, et al., 2006), TrichDB.org, for *Trichomonas* species (Aurrecoechea, et al., 2009), GiardiaDB.org for *Giardia* species (Aurrecoechea et al., 2009), TriTrypDB.org, for the *Kinetoplastid* species and ToxoDB.org for *Toxoplasma* and *Neospora* (Kissinger et al., 2003).

In EuPathDB, the component sites have much in common. They are all built on three building blocks: The Genomics Unified Schema, The Web Development Kit, and The Web Service Framework.

The Genomics Unified Schema (GUS) is a database schema designed for bioinformatics data storage. GUS relationships were designed to follow an ontology of actual biological relationships. It defines tables and relationships that make the setup and implementation of this type of system easier (Davidson et al., 2000).

The Web Development Kit (WDK) was developed as a common set of functions that build a system on top of GUS. It allows for questions and queries to be described in a "model" structure to configure the WDK to work with any schema (Aurrecoechea et al., 2007).

The Web Service Framework (WSF) is a generic Web service framework (Wang et al., 2009). The WsfService contains only one operation, invoke(), which loads a plug-in that is passed as a parameter, and executes the logic in the plug-in. The return type of the WsfService is a WsfResponse which contains two fields: a message used to return information about the invocation and a two dimensional array of strings

containing the results of the query. The federation services are implemented as plug-ins to WsfService.

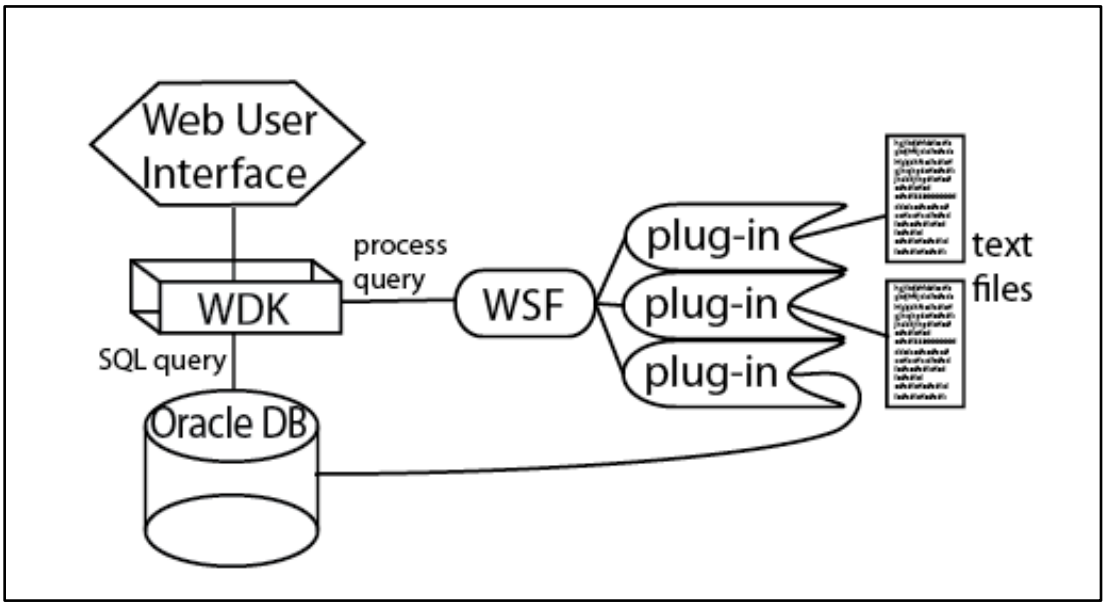

Figure 3 Primary Key Query Architecture At a Component Site

Figure 3 demonstrates how these different parts of the system work together to create a working website. As you can see, the WDK has two distinct ways to collect information for answers to questions. The first method is a direct access to a SQL database; here Oracle is used. The second method is to use a Web service in the form of the WSF, to access data that typically reside in a component database. In some cases the data is stored by other means, such as files or XML documents.

The component databases that are part of the EuPathDB system are all implemented and maintained by the EuPathDB team. Though all of the sites are developed atop the same architecture and use the same schema, the underlying databases still have differences in the way that the data is represented within the databases. This means that the SQL for a given question can differ amongst the

component sites. For example, PlasmoDB and ToxoDB both include the GenesByMetabolicPathway query, but they use different queries to retrieve the answer. The SQL statements can be found in figures 4 and 5, respectively.

```
select * from (
SELECT gf.source_id, '@PROJECT_ID@' as project_id,
apidb.tab_to_string(CAST(COLLECT(distinct decode(dr.lowercase_secondary_identifier, null,
dr.primary_identifier,dr.lowercase_secondary_identifier)) AS apidb.varchartab), ', ') as met_pathways
FROM apidb.geneattributes gf, dots.DbRefNaFeature drnf, sres.DbRef dr,
(SELECT idrnf.na_feature_id
 FROM dots.DbRefNaFeature idrnf, sres.DbRef idr
 WHERE idr.primary_identifier = $$metabolic_pathway$$
 AND idrnf.db_ref_id = idr.db_ref_id) internal
 WHERE internal.na_feature_id = drnf.na_feature_id
 AND gf.na_feature_id = drnf.na_feature_id
 AND drnf.db_ref_id = dr.db_ref_id
 group by gf.source_id)
```

Figure 4: PlasmoDB SQL Listing for GenesByMetabolicPathways

```
Select * from (
SELECT gf.source_id, '@PROJECT_ID@' as project_id,
apidb.tab_to_string(CAST(COLLECT(distinct decode(dr.lowercase_secondary_identifier, null,
dr.primary_identifier,dr.lowercase_secondary_identifier)) AS apidb.varchartab), ', ') as met_pathways
FROM apidb.geneattributes gf, dots.Transcript t,dots.DbRefaaFeature draf, sres.DbRef dr,
dots.TranslatedAaFeature taf,
(SELECT idraf.aa_feature_id
 FROM dots.DbRefaaFeature idraf, sres.DbRef idr
 WHERE idr.primary_identifier = $$metabolic_pathway$$
 AND idraf.db_ref_id = idr.db_ref_id) internal
 WHERE internal.aa_feature_id = draf.aa_feature_id
 AND gf.na_feature_id = t.parent_id
 AND t.na_feature_id = taf.na_feature_id
 AND taf.aa_feature_id = draf.aa_feature_id
 AND draf.db_ref_id = dr.db_ref_id
 GROUP BY gf.source_id)
```

Figure 5: ToxoDB SQL Listing for GenesByMetabolicPathways

This type of heterogeneity is the type of problem that the Web service federation was designed to solve.

3.2 History of EuPathDB and Problems

EuPathDB's initial release (previously called ApiDB) offered access to CryptoDB.org, PlasmoDB.org, and ToxoDB.org and the communication was implemented using database-links. This was a functional implementation as it allowed communication at the database level. However, this solution proved to be problematic when other organisms were added. *Trichomonas vaginalis* and *Giardia Lamblia* were added with the sites TrichDB.org and GiardiaDB.org, respectively. In order to add these sites to the portal, the SQL statements had to be altered for each and every question in the portal. Differences in the SQL queries came about because the data for each component site are slightly different and are represented in slightly different manners. Though every attempt is made to keep the data uniform, the data itself causes this type of heterogeneity. With more organisms to be added to the project in the following year, it became clear that an improvement was needed to make the portal scalable.

Another requirement for this new system is to ease the maintenance of the portal. New data are being created daily and then loaded into the databases for upcoming releases of the component sites. Whenever a new query is added to the site or an existing query is updated to fix an issue or add some new functionality, the portal must be updated as well. Since the data loading team is located mostly at the University of Pennsylvania and the portal team is at the University of Georgia, the communication about these types of changes is imperfect. This leads to unexpected outages of the portal and nearly constant maintenance edits to the SQL supporting EuPathDB.

To this end, we implemented a federation using Web services to link the sites using a common, stable, abstract interface to allow access to all queries on all component sites.

Chapter 4

Architecture

4.1 Web Services Federation

Using Web services for federation involves providing a public interface for private and local databases. A Web service federation system consists of a federation service and component services. The federation service is designed and implemented by the administrators of the federation. The federation service defines the federation query language, dissects queries into pieces that are sent on to the appropriate component systems for processing and composes the results from component systems into a coherent whole for return to the user. The federation service also defines an interface that component administrators must adhere to in order to join the federation, the component interface. It is important that the component interface possesses two characteristics: stability and abstraction. The component interface must be stable because the federating service relies on it for communication with the component service. The interface should be robust and capable of handling features of any system that it might by abstracting. Any changes to this interface will require updates to all systems within the federation. The interface should provide abstraction of the detailed differences amongst the component systems, making access consistent. Absolute uniformity is not required, but the more the component interfaces vary and change, the more maintenance is required to keep the system functioning.

Still, the question remains of how to use Web services to provide abstraction from the specific interfaces of the systems in the federation. I will now explain five approaches and explore the pros and cons for each option.

Table 1 Approaches to implementing Web service integration

	Abstraction	# Services	# Operations / Services	# Queries	Discoverable	Maintenance
1	**Direct SQL**	1	1	∞	Yes	Low
2	**Many Services**	∞	1	∞	Yes	High
3	**Generic Service**	1	1	∞	No	Low
4	**Generic Service + Metadata**	1	2	∞	Yes	Low
5	**Categorized Services**	∞	1	∞	Yes	Automated

The most obvious solution is to implement a Web service as a single operation. The operation would supply the connection information to the database and would accept a SQL statement as input. This service can be described by a WSDL specification, published, and discovered. While this approach is simple to implement, it has two flaws in our application design. The first flaw is a lack of abstraction. In order for the federating system to directly send SQL queries to the service, it would have to know the schema of the component databases. Thus, the federating system and the local system would be tightly coupled and unable to exercise design autonomy. The second and more serious flaw is a matter of security. If we allow anything that resembles a SQL query to be executed on the database, the system is open to malicious SQL queries that could damage or destroy the database. Some parsing could be done to limit this type of breach, but it is difficult to eliminate completely.

The second solution is to create one Web service for each query. Each service would contain a single operation specific to a certain query. Input to the service would be parameters that are inserted into the query to give specific results. As before, each service would have a WSDL file that describes the service's functionality. These WSDL files could be annotated using the SAWSDL attributes to add semantics. This significantly lessens the security threat by making the inputs a smaller set of terms, and thus validating them becomes simpler and more precise. However, it would be difficult to maintain such a large set of Web services. In the case of EuPathDB, there are over 80 queries and any changes to the parameters of a query would require changes to the respective service, thus increasing the maintenance issues for both the component and the federation administrators. A similar option would have only one service with an operation for each query. Here there would be a single service will over 80 operations. This has the same advantages and disadvantages as having one service per question.

The third solution is to implement a generic Web service. This service provides only a framework for invoking “plugins” that are defined according to a given API. These plugins are classes that contain the business logic for the service. The generic service itself offers no logic and no useful function; but together with its plug-ins, it is a powerful, robust, easily implemented solution to building web services. Since any invocation of the service can ask for any available plug-in, the applications for this particular service are limitless.

This service would contain one operation, “invoke”. The “invoke” operation will do all of the work for the service. It will accept four parameters as input: pluginName, queryName, input, and output. The pluginName parameter is critical to this

implementation. Since the service itself holds no logic for performing queries, then a plug-in must be defined to tell the service what the user wants accomplished by this call. The remaining parameters are plug-in specific and only hold meaning for the plug-in that was asked for. This generic service provides an outside interface that is very stable. Since no business logic is in the service, and the parameters are all abstracted into a container, the interface should rarely need to be altered. A generic service is the most abstract of the options. The lack of business logic in the service means that the same service could be implemented on multiple databases. The drawback to this implementation is that it is not describable in a meaningful way via a WSDL. Thus it cannot be published and cannot describe itself in sufficient detail to be generally invoked. A client must have knowledge of the plugins and the inputs to those plugins to invoke this service.

Option four solves the problems with option three. It provides another operation to the service to describe the inputs necessary to utilize the *invoke* operation. The new operation is describable and discoverable via a WSDL file. With this operation in place, a client can gather all the information needed to utilize the generic service. This also offers the flexibility that the operations and plugins available to the generic service can change and the client will be made aware of those changes in near real time.

The final option combines the pros of options 2 and 3. It utilizes the flexibility of the generic service and standardizes the interfaces by utilizing code generation from a file describing the questions in the system. The final result of this implementation is a generic service that is used by the federation system where the plugins can be written to conform to the needs of the federation. A generator is written to parse a

configuration file to create a wrapper for the generic service, which takes specific inputs and outputs. These generated services can be described with WSDL files, semantically annotated and published to a registry.

4.2 Case Study: EuPathDB Architecture

As mentioned earlier, the EuPathDB sites utilize a common infrastructure consisting of GUS, WDK and WSF. This infrastructure provided a useful backdrop against which a Web service federation system could be implemented. The WDK contains a "model". The model is a hierarchy of documents that defines all queries, parameters, and controlled vocabularies used in the system. The model is common to all sites but contains some attributes that are site specific. By leveraging the model information through the WDK, the implementation of the Generic Service approach to Web service integration (Table 1, option 3) was simplified considerably. Yet, because of choosing option 3, the services are internal to the sites only because the inputs and outputs are not describable in a WSDL. In the WDK, the queries are posed in the form of questions. Most questions allow the user to choose the organism, or organisms, that they wish to explore, and then provide values to parameters that are specific to the question they are asking. The question is translated by the WDK to a query. The query is either of type "sqlQuery", which contains a SQL statement that is parsed with the parameters and executed, or "processQuery", which uses the WSF to execute a Web Service to get the results of the query. Once finished, the results are returned in the form of an answer to the originally posed question. We utilized the WSF and the "plugins" architecture because of its easy integration in to the WDK to implement the services that would act as our federation service, ApiFedPlugin, and component

service, WdkQueryPlugin. In EuPathDB.org, all queries are “processQuery” that point to ApiFedPlugin. This plug-in decomposes the query and passes it to the WdkQueryPlugin on the relevant component sites. Once all component invocations return data, then ApiFedPlugin compiles the data into a single result that is returned to the user.

ApiFedPlugin

ApiFedPlugin does the work described for the federation Web service. The system uses the Invoke operation to get results for any question. The inputs are sent as an array called the parameter list. Invoke takes the parameter list and finds the organism parameter. This parameter determines which component sites need to be included in this query. In the absence of an organism parameter, the query is sent to all component sites. The parameter list is parsed to be sure that component specific parameter values are not sent to the wrong site. For example, CryptoDB.org will produce an error if “*Plasmodium falciparum*” is sent as input to the organism parameter. For each component, a thread is spawned and an invocation of the WSFService on the component site is made using the plug-in WdkQueryPlugin.

Once all invocations have returned, the results are combined into one 2-dimensional array of strings with the columns from each invocation matching to ensure a semantically meaningful result. This object is inserted in a WsfReponse object along with any messages that were returned by the component sites. That object is returned to the WDK to continue processing.

WdkQueryPlugin

The WdkQueryPlugin resides on each of the component sites. It works as a generic container for the queries that are sent from ApiFedPlugin. WdkQueryPlugin is configured on each site to initialize with the model.xml for that site. Once initialized, the plug-in is stored in memory to avoid the expensive task of repeatedly parsing the XML model file. When invoked, WdkQueryPlugin retrieves the query from the model, validates the parameters and executes the query. The results are put into a WsfReponse object and returned to the ApiFedPlugin.

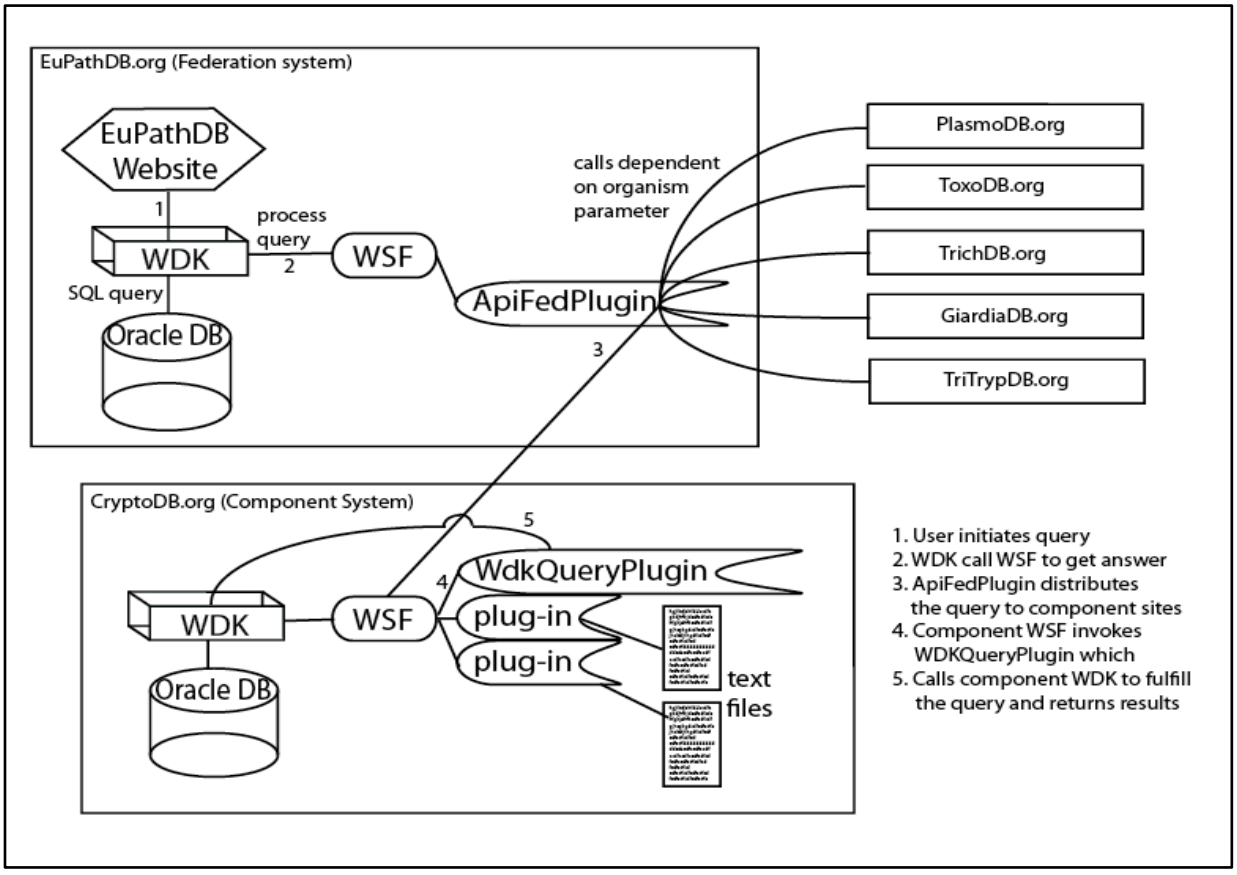

Figure 6 Architecture of EuPathDB Federation

Figure 6 shows the path through the system graphically. As an example, if a user selects the GenesByGeneType question and executes it with the following parameters:

organism = *Cryptosporidium parvum,Plasmodium vivax*
gene_type = protein_coding
pseudogenes = No

The EuPathDB model instructs WDK to send this query to the WSF with the ApiFedPlugin for processing. Once there, ApiFedPlugin spawns two threads based on the organism parameter.

Thread #1 : organism = Cryptosporidium parvum
gene_type = protein_coding
pseudogenes = No
queryName = GeneQuestions.GenesByGeneType
Thread #2 : organism = Plasmodium vivax
gene_type = protein_coding
pseudogenes = No
queryName = GeneQuestions.GenesByGeneType

Thread #1 invokes the WdkQueryPlugin on CryptoDB.org and Thread #2 invokes the WdkQueryPlugin on PlasmoDB.org. Each component service utilizes the WDK to find the given query and execute it using the given parameters. All validation of inputs is done by the component WDK. The WdkQueryPlugin returns the results to ApiFedPlugin which combines the component results into a single answer and returns it to EuPathDB's WDK for display.

Results of EuPathDB Web Service Federation

Imagine for a moment what it would take to gain access to data about eukaryotic pathogens. In the past, to compare results between *Plasmodium* and *Cryptosporidium*

would require going to two different sites, running two different queries, and examining two lists of results. With the old implementation of the EuPathDB.org, the user was given a single point of access, but was frustrated by inconsistent results and down time of the site. Given the new implementation of the federation using Web services, EuPathDB.org has solved all of these problems. All questions from all component sites are available on the portal. The user can even execute orthology questions on their results to find comparable genes in other species.

This system delivers solutions to two major issues around database federation. Autonomy of design and maintenance are important to the individuals maintaining the component sites. Web service federation maintains this feature by abstracting all of the local decisions behind the Web service interface. By using the generic Web service approach, we ensure that the interface to this service will change rarely, if ever. The component managers implement the component plug-in to allow as much or as little access to their data as they see fit. The federation has no control over the component site other than the definition of the generic Web service interface. The second major issue is one of heterogeneity. Structural heterogeneity is abstracted away by the Web services. Because the interface is not connected to the database in any way, the schema is hidden from the federation system. Data heterogeneity is a more difficult problem. In the EuPathDB project, this is less of a problem due to the fact that all of the component sites are bioinformatics sites yielding the same type of genomic and proteomic data. This is not to say that we did not face data heterogeneity. In several cases, sacrifices were made by some component site to use the naming conventions of another in order to bring this system online in a timely fashion. In a more robust sense,

data heterogeneity can be addressed by the use of semantics. An advantage of this system is that all of the data matching and handling of data heterogeneity is done by the federation system. The component systems have no need to change or implement anything differently. This information would allow for disparate systems to be federated as long as they agreed upon ontology to define their data.

This Web service federation made EuPathDB a more feasible project. We were able to make the project scale to more organisms and more sites. We were able to decrease the amount of maintenance the portal required to remain functioning. We were able to make the portal a more stable site that is useful to the research community. Under the original implementation, adding a site took weeks. First DB-links had to be setup for the new database. Then SQL statements for each query had to be added to the SQL statements of the portal and debugged. With the Web services, this process now takes only a few hours. The component site has WSF and the component plugin, WdkQueryPlugin, in place from the build process. Setup of the federation system means editing the configuration files for both plugins. The component service needs to know where to find the model file and were the component site is installed. The federation service must be given the URL of the component service, the name of the component site and a regular expression used to parse the organisms to route only the appropriate questions to the new component site. After this, the application server reloads the portal and the component sites and the federation now services a new site. With this system in place, EuPathDB has recently added GiardiaDB.org, TrichDB.org and TriTrypDB.org. EuPathDB also had its contract renewed by NIH, partially due to its ability to scale.

Before the services-based federation system, EuPathDB required almost constant maintenance. Since the component sites are autonomous entities, they are on their own release schedule. Each time a component site releases a new version, EuPathDB had to be updated. This was not manageable with three sites and threatened to make the project unmanageable if more sites were added. With the federation system, this is no longer a concern. Once a site is configured as part of the federation, the Web services handle the communication with the site. Each new version of a component database no longer means an update of the portal. The Web services dynamically pick up the new data when a question is requested from the new component. If one day that component is version 2.3 and the next it is version 2.4 or 3.0, everything continues to function. Table 2 compares the time needed to perform typical maintenance and expanding tasks before and after the federation system was put in place.

Table 2: Maintenance Comparison for EuPathDB Project

Problem	DB-Links	Time	WS Federation	Time
Adding new Component Site	Cut all SQL from component and paste in as UNION in portal	2 weeks per site	Configure Federation Service with URL of new component Web Service	30 minutes
Component site changes schema	Cut and paste new SQL into portal SQL	2 hours per query	Handled by abstraction	none
Component site modifies SQL	Cut and paste new SQL into portal SQL	2 hours per query	Handled by abstraction	none

The Web service federation also increased the stability of the portal site. Under the old system, a change to a component system could bring down EuPathDB. If a column was renamed or a parameter value changed, then an update had to be made to the portal. This meant downtime for the portal. The new system abstracts such changes so that they do not affect the portal. The component systems retain total autonomy over their systems and most changes do not require downtime for EuPathDB.org.

In addition to the benefits needed for a federation, this implementation offers other benefits. One of these benefits is the ability to implement public Web services to make all of the data available to the EuPathDB system programmatically accessible. This enables researchers with programming skills to process large amounts of data quickly and easily. It also will enable EuPathDB to collaborate with other Bioinformatics Research Centers to share data. An example of this is represented by our plans to implement Web services to enable EuPathDB to share functional proteomics data to aid the annotators at GeneDB. In return, GeneDB will provide services to allow EuPathDB to gain real-time access to their annotations.

4.3 Incorporation of Outside Databases

With the Web service federation system in place, it becomes a building block on which a greater system can be implemented. The federation administrator can add other systems to the federation with little or no effort. Once the component site has implemented a service conforming to the federation interface then communication can occur. But this does not solve all of the problems. Differing semantics amongst the federation and component sites can create problems for the federation as it attempts to

incorporate results from the new component into results sets. Ideally, the federation system should become smarter in order to handle the range of data that could come from outside services. To this end, the federation system may add the functionality of ontological data mediation (Nagarajan et al., 2007) to its arsenal. This approach will allow the federation and constituent services to communicate efficiently even if they are using differing terms and data structures. By using SAWSDL to annotate the federation and component WSDL's input and output elements with ontological concepts and mapping functions, the services can have the information needed to successfully map terms from one system and ontology to the another (Nagarajan et al., 2007).

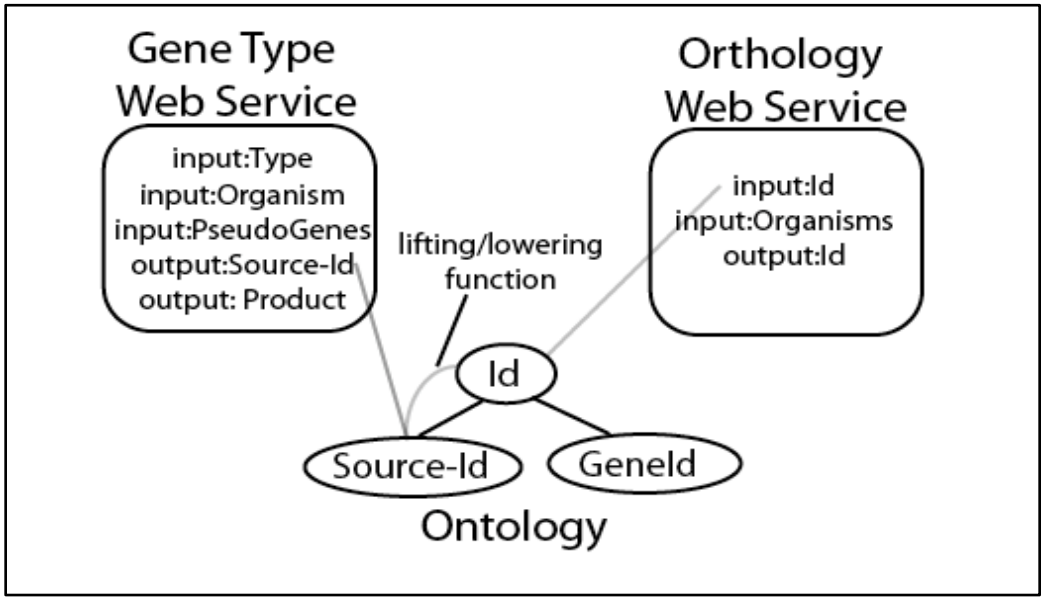

Figure 7: Data Mediation

As an example, presently EuPathDB is integrating six component sites. Implementing this federation with Web services will allow EuPathDB to reach outside it current component sites and include systems that are maintained by other

organizations. Work is ongoing to have EuPathDB Web services communicating with other services outside the EuPathDB group. One such collaborator is the Sanger Institute that maintains the GeneDB web site. Once complete, this collaboration will allow GeneDB annotators to search EuPathDB for evidence to support their annotations of genes on the *Plasmodium* genome. EuPathDB will gain access to the latest annotations of the genome in real-time and thus always provide the most up-to-date information to the user base.

For these types of advances to be possible, EuPathDB will need to add publicly available Web services to its complement of internal services. This task is simplified by the existence of the internal services. Our implementation of these public services will be to put a wrapper around the generic service that is used for the federation. The wrapper services will be a lightweight operation that provides standards-compliant WSDL that can be invoked with the parameters needed for that query. Once the service receives the inputs, it will parse them into a form recognized by the WSF and pass them along to be handled by the federation system services. Once the results are returned, the same operation is performed in reverse and the output is put into a standard format to be sent to and consumed by the client.

4.4 Supporting Workflows

Research is ongoing to compose Web services into processes. Languages such as the Web Service Business Process Execution Language (WS-BPEL) define these processes (Weerawarana et al., 2005). By implementing a federation using Web services, the components could be made available to these processes through the

component service. The federation as a whole could also be included in such a process.

Current work at EuPathDB is moving toward such a goal. EuPathDB has developed a new user interface for all of its websites, known as the “Strategy system”. This interface allows users to build very complex queries using Boolean and transformation operations. Figure 7 shows a screen shot of this new interface. The Boolean operations are clearly visible, easy to understand, and intuitive to use. User feedback on this system has been overwhelmingly positive because it has brought the ability to combine data sets to the forefront of the site. These strategies can be saved and retrieved on the website for later viewing and editing. Making this system even more powerful is the ability to nest the strategies within one another and create “sub-strategies” within a step of a strategy.

With the Strategy system and the Web service federation in place, the next step is to combine the two systems to create a powerful workflow system. The design is to allow users to create a strategy and download an XML representation of their strategy. The XML is then passed as input to a Web service to get the results of that strategy. This service will allow users to create a workflow and programmatically manipulate the input and outputs to gain some information about the data that would have been time consuming and error-prone to do by hand.

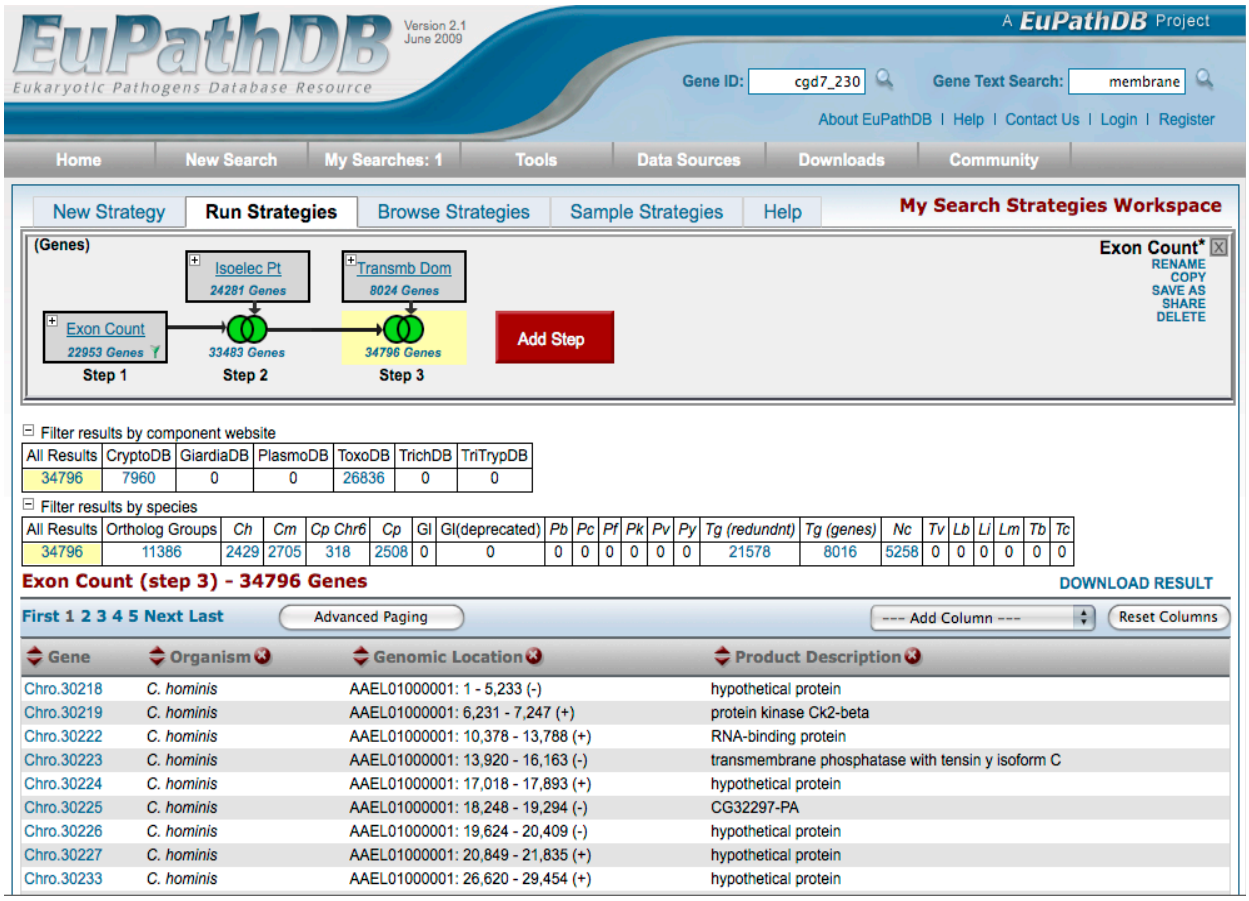

Figure 8: Search Strategy System on EuPathDB.org

Chapter 5

Evaluation

As mentioned before, three features of database federations are heterogeneity, distribution, and autonomy. Web services offer solutions to each of these issues. Autonomy and structural heterogeneity are handled by the abstraction of the underlying data source that Web services offer. Web services can offer high-level abstraction of the underlying source and thus render the need for low-level knowledge of that system unnecessary. Distribution is handled by the fact that the services are available over the Web using standard protocols. Semantic heterogeneity is handled by the annotation of the services from an agreed upon published ontology. Agreement on terms ensures that results from disparate component systems can be assembled by the federation service into a single coherent result.

The grounding principle of federated databases is that one can integrate several systems without having to alter or discard the constituent systems. Neither should it require current management to relinquish local control of their system. The technology used to implement this system must allow communication in such a fashion that these goals are achieved. Web services can supply a means to this end, since they can provide both the autonomy and robustness needed for this system. Web services allow for systems to be added to the federation with no changes whatsoever. The local developers must get the interface from the federation system and implement a Web service that adheres to that public interface. Once this is done, these same local

developers are responsible for defining the logic for this service to access their system. This logic is not of concern to the federation and is not defined by the federation. This allows for any data source that has data to be added to the federation with minimal work by the federation administrators and no modifications to the component data source.

Roughly speaking, Web services can be thought of as remote procedure calls. In this sense, the service is the application and the operation is the procedure. It has inputs that are defined in the WSDL to be of a certain type and structure. These inputs are akin to the parameters of a procedure. It has outputs that are also defined in the WSDL that are akin to the return value of the procedure. Using this analogy, it is easy to see how a Web service can provide absolute abstraction of the system to which it is providing access. Because of this abstraction, the administrators of the constituent systems maintain complete control over the implementation of their system. Web services offer another type of autonomy as well. The constituent system must implement the service in such a way that it is accessible by the federation system, but other variables about the implementation of the service are left up to the local developers. Since Web services use SOAP over HTTP to communicate, there are no restrictions on the programming language, SOAP engine, or any other component of the system that is regulated by the federating administrators. Thus each constituent system retains total autonomy over its data and services while it is involved in the federation.

Semantics is the study of meaning. Meaning is central to a federation because if the federation is using a term in one sense and the component is using the same term to mean something else, then incorrect results could be returned for a query. As mentioned, IBM and the University of Georgia developed WSDL-S which is WSDL

annotated with terms from ontology to provide meaning to the functionality inputs and outputs of each operation of a Web service (Akkiraju et al., 2005; Miller et al., 2004). WSDL-S has since evolved in a lighter weight version of itself that is a current W3C standard called SAWSDL (Kopecky, et al., 2007). With these annotations in place the federation and component systems only have to agree on the ontology to reduce semantic ambiguity.

Web services provide advantages over other federation implementations. Using Web services for federation creates a dynamic federation. It is dynamic in the sense that constituents can come and go without degrading the federation. This dynamic quality adds a great deal of functionality to the federation. First, a constituent system could suffer a failure and not be available to answer a query. This system can leave the federation without causing any lapse in functionality. Of course some data is no longer available when a constituent leaves the system, but the federation will still function.

Web services offer another advantage; they are programmatically accessible from outside the federation. If the federation service is published, then any of the constituent services are also published and are accessible. A federation can be viewed as a “star topology” or a centralized access point. These design patterns mean that the federation system, acting like the access point, is known to the component systems, acting as the nodes, but the nodes do not know about each other. The publishing of the federation service gives each node the ability to share and retrieve data from each of the other nodes.

Web service federation has another advantage, almost a side effect. Because of the nature of Web services, using this technology yields a system that is dynamic. In

order for a system to enter into the federation, the schema mapping must exist, so if the system leaves the federation it can return at any time without any further modifications (assuming nothing was changed during its time away). In much the same way, if the Web service is defined and working correctly, the administrator of one site can disconnect from the other site physically, yet the mechanism would be in place to reestablish communication at any time in the future. Over time, this process would develop a list of candidate databases that could join the federation at any given time; thus creating a truly dynamic database federation.

Chapter 6

Conclusions and Future Work

Web services are a viable and flexible means to implement a database federation due to the high level of abstraction the architecture provides. Federations require a level of abstraction and uniformity to provide the user with consistent access to data. By using Web services, data can be abstracted by the use of a common vocabulary (in the case of EuPathDB, this is currently given in an XML specification) or the use of semantic techniques such as ontologies (which EuPathDB plans to implement in future releases). These allow for the mapping of terms to ensure that the terms used in all systems are consistent. Federations also require the structure of the component systems to be abstracted so that the federating system can execute queries. Web services provide this type of abstraction as well. Web services abstract the structure of the component databases. The interface to the service is all the federation must be aware of to invoke a query on a constituent system. The management system and query language are of no concern and thus translation at the federation level is not necessary.

These techniques and technologies have been implemented successfully on the EuPathDB project. The system has aided the developers and administrators of the EuPathDB portal a great deal by significantly decreasing the time needed to maintain the current site and to add new component sites to the project. The Web service federation has also cut down on errors in the federation by abstracting all SQL and

other database query languages out of the portal configuration. This has had a great effect on the maintainability of the portal. In the past small changes in SQL statements or schema updates, would cause the portal to have to be updated and re-released. With the abstraction that the Web services provide, these changes no longer affect EuPathDB.org. Expandability has also been improved as a result of the Web service federation. The addition of new systems now takes developers only minutes to accomplish what took days under the old implementation. Due to time and implementation constraints, the EuPathDB system is not perfect, and some problems do occur. But these problems now affect only a single component site while the remainder of the portal continues to function correctly.

The use of Web services to implement a database federation creates a platform on which a workflow system can be constructed. For EuPathDB, which uses only predefined SQL statements, this allows the users to combine queries to create custom queries that are useful to their work. They can then execute these queries against the Web services to effectively run a custom built query. This custom query, or process, is executed as a workflow against the Web services that represent each of the questions within the query.

This current system will prove to be a stepping-stone for many new features in the EuPathDB project. The team is now working to define the requirements for automatically generating publicly accessible Web services based on the generic service that is used for the federation. This was described earlier in this thesis as Categorized Services (table 1, option 5). The public services can be annotated with SAWSDL

allowing them to handle more diverse data mediation tasks and add a greater range of diversity in the systems that join the federation.

BIBLIOGRAPHY

Aggarwal, R., Verma, K., Miller, J. A., and Milnor, W., (2004). *Constraint Driven Web Service Composition in METEOR-S,".* Paper presented at the Proceedings of the 2004 IEEE International Conference on Services Computing, Shanghi, China 23-30.

Akkiraju, R., Farrell, J., Miller, J. A., Nagarajan, M., Schmidt, M., Sheth, A., Verma, K., (2005). Web Service Semantics - WSDL-S: A Joint UGA-IBM Technical Note. *IBM AlphaWorks Technical Note, version 1.0*, 42.

Aurrecoechea, C., Brestelli, J., Brunk, B. P., Carlton, J. M., Dommer, J., Fischer, S. (2009). GiardiaDB and TrichDB: integrated genomic resources for the eukaryotic protist pathogens Giardia lamblia and Trichomonas vaginalis. *Nucleic Acids Res, 37*(Database issue), D526-530.

Aurrecoechea, C., Brestelli, J., Brunk, B. P., Dommer, J., Fischer, S., Gajria, B. (2009). PlasmoDB: a functional genomic database for malaria parasites. *Nucleic Acids Res, 37*(Database issue), D539-543.

Aurrecoechea, C., Heiges, M., Wang, H., Wang, Z., Fischer, S., Rhodes, P. (2007). ApiDB: Integrated Resources for the Apicomplexan Bioinformatics Resource Center *Nucleic Acids Res, 35*(Database issue), D427-D430.

Curbera, M., Khalaf, R., Nagy, W., Mukhi, N., Weerawarana, S. (2002). Unraveling the Web services web: an introduction to SOAP, WSDL, and UDDI. *Internet Computing, IEEE, 6*(2), 7.

Giardine, B., Riemer, C., Hardison, R. C., Burhans, R., Elnitski, L., Shah, P. (2005). Galaxy: a platform for interactive large-scale genome analysis. *Genome Res, 15*(10), 1451-1455.

Haas, H. and Brown, A. (2004) W3C Working Group Note 11 Febuary 2004. http://www.w3.org/TR/ws-gloss/.

Goto, M. K. (2000). KEGG: Kyoto Encyclopedia of Genes and Genomes. *Nucleic Acids Ressearch, 28*(1), 27-30.

Gruber, T. R. (1991). The Role of Common Ontology in Achieving Sharable, Reusable Knowledge Bases. In J. A. Allen, R. Fikes, & E. Sandewall (Eds.), Principles of Knowledge Representation and Reasoning: Proceedings of the Second International Conference, Cambridge, MA, 601-602.

Hadley, M., Sandoz, P., *JAX-RS: Java API for RESTful Web Services*. Revision 1.1 Editor's Draft Aug. 21. https://jsr311.dev.java.net/drafts/spec20090821.pdf

Hammer, D. and McLeod, D. (1980) *A Federated Architecture for Database Systems*. Proceedings of the May 19-22, 1980, National Computer Conference. (database management seesion) 283-289.

Heiges, M., Wang, H., Robinson, E., Aurrecoechea, C., Gao, X., Kaluskar, N. (2006). CryptoDB: a Cryptosporidium bioinformatics resource update. *Nucleic Acids Res, 34* (Database issue), D419-422.

Miller, J. A., Verma, K., Rajasekaran, P., Sheth, A., Aggarwal, R. and Sivashanmugan, K. (2004). WSDL-S: Adding Semantics to WSDL - White Paper. *Technical Report*(UGA-CS-LSDIS-TR-04-011), 44.

Kissinger, J. C., Gajria, B., Li, L., Paulsen, I. T., and Roos, D. S. (2003). ToxoDB: accessing the Toxoplasma gondii genome. *Nucleic Acids Res, 31*(1), 234-236.

Kopecky, J. V., T. Bournez, C. Farrell, J. (2007). SAWSDL: Semantic Annotations for WSDL and XML Schema. *Internet Computing, IEEE, 11*(6), 7.

Li, K., Verma, K., Mulye, R., Rabbani R., Miller, J. A., Sheth, A. (2004). *Designing Semantic Web Processes: The WSDL-S Approach*. In Jorge Cardoso and Amit P. Sheth, editors, 'Semantic Web Services, Processes and Applications', Vol.3 of Semantic Web And Beyond Computing for Human Experience, Springer, 2006. 161–193.

Nagarajan, M., Verma, K., Sheth, A., Miller, J. A. (2007). Ontology Driven Data Mediation in Web Services. *International Journal of Web Services Research, 4*(4), 22.

Sayers, E. W., Barrett, T., Benson, D. A., Bryant, S. H., Canese, K., Chetvernin, V. (2009). Database resources of the National Center for Biotechnology Information. *Nucleic Acids Res, 37*(Database issue), D5-15.

Sheth and Larson (1990). Federated database systems for managing distributed, heterogenous, and autonomous databases. *ACM Computing Surveys, 22*(3), 53.

Smedley, D., Haider, S., Ballester, B., Holland, R., London, D., Thorisson, G. (2009). BioMart--biological queries made easy. *BMC Genomics, 10*, 22.

Verma, K., Gomadam, K., Sheth, A., Miller, J. A. and Wu, Z. (2005). The METEOR-S Approach for Configuring and Executing Dynamic Web Processes," 1-34

Wang, Z., Gao, X., He, C., Miller, J. A., Kissinger, J. C., Heiges, M., Aurrecoechea, C., Kraemer, E. and Pennington, C. (2009). An Evaluation of Multiple Approaches for Federating Biological Data. *Journal of Information Technology Research, 2*(2), 22.

Weerawarana S., Curbera, F., Leymann, F., Storey, T., Ferguson, D. F. (2005), *Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging and More*, Prentice Hall PTR, Upper Saddle River, NJ, 2005

Wilkinson, M. D., Links, M. (2002). BioMOBY: An open source biological web services proposal. *Briefings in Bioinformatics 2002, 3*(4), 10.

Wilkinson, M. D., Senger, M., Kawas, E., Bruskiewich, R., Gouzy, J., Noirot, C. (2008). Interoperability with Moby 1.0--it's better than sharing your toothbrush! *Brief Bioinform, 9*(3), 220-231.

Yang, B., T. X., Zhao, J., Kommidi, C., Soneja, J., Li, J., Will, R., Sharp, B., Kenyon, R., Crasta, O., Sobral, B. (2006). *Bioinformatics Web Services*. Paper presented at the 2006 International Conference on Bioinformatics.

APPENDIX A

IMPLEMENTATION DETAILS OF EUPATHDB WEB SERVICES

The EuPathDB Federation system is comprised of two Web services. These services were implemented within the framework of a single, generic service known as the Web Service Framework, or WSF. The WSF is a service that contains only the logic for accepting invocations, executing a plug-in through reflection, and returning results. It contains no business logic code and does no real work toward a useful goal.

```
1  public WsfResult invokeEx(
2     String pluginClassName, String projectId, String[] paramValues,
3     String[] columns) throws ServiceException {
4        int resultSize = 0;
5        Map<String, String> params = convertParams(paramValues);
6        IWsfPlugin plugin;
7        if (plugins.containsKey(pluginClassName)) {
8          plugin = plugins.get(pluginClassName);
9      } else {
10         Class<?> pluginClass = Class.forName(pluginClassName);
11         plugin = (IWsfPlugin) pluginClass.newInstance();
12         plugin.setLogger(Logger.getLogger(pluginClass));
13         plugins.put(pluginClassName, plugin);
14    }
15    WsfResult result = plugin.invoke(projectId, params, columns);
16    resultSize = result.getResult().length;
17    prepareResult(result);
18    return result;
19  }
```

Figure 9: Listing of the InvokeEx() function

Above is a listing of the InvokeEx() function from the WsfService Class. This function is responsible for handling the invocation request to the WSF. It is important to note here that a pluginClassName is given as a parameter. That parameter if used in line 10 to

create an instance of that class. This is the plug-in class which contains the business logic for this invocation of the WsfService. The services for the EuPathDB project federation are implemented as this type of plug-in.

The WsfPlugin class is another integral part of this system. It defines an abstract class with the execute() function. This function must be implemented by all plugins to the WsfSerice. This is the function where the business logic for the service is implemented. Below is a listing of the interface for the execute function.

```
1   protected abstract WsfResult execute(String queryName,
2       Map<String, String> params, String[] orderedColumns)
3       throws WsfServiceException;
```

Figure 10: Listing of execute() function

The WdkQueryPlugin is the service that is running on each of the constituent databases. It is responsible for doing the work of the federation. The inputs to the execute function of this plugin are:

- QueryName – Name of the query that is to be executed on this component site
- Params – Map of term/value pairs defining the parameters of the query to be run
- orderedColumns – Array of the columns to be returned by this invocation

WdkQueryPlugin is configured with the knowledge of the sites WDK. Thus it has an in memory copy of the model. It searches this model to determine if the requested query exists on this site. If it is found, then execution is turned over to the WDK to execute the query and return the results. Once the WDK returns to the plug-in, the results are parsed and any columns that were not requested are removed from the result set. A message is also formatted that contains the number of results returned by

this invocation. If the query is not found, the service returns a “-2” value in the message to the federation signifying that the query was not available on this site. In the case that there is an error on the component site, the WdkQueryPlugin will return “-1”.

The ApiFedPlugin is the federation service in this system. It is implemented with the same inputs and output as the component plugin, yet it contains more logic. ApiFedPlugin utilizes the fact that most queries in the EuPathDB system have ”organism” parameters. This parameter is used to parse the query and determine which component sites need to be accessed for this particular invocation. This is accomplished by the use of regular expressions that ApiFedPlugin is configured with at invocation time that defined what organisms go to which component site. Once the sites are determined, ApiFedPlugin spawns a new thread for each invocation. This allows the system to achieve a performance that is equal to that of the slowest component site rather to the sum of all sites. Once all results are returned, ApiFedPlugin will parse the results and combine them into a single result set that is returned to the WDK for display. It also combines all of the return messages to inform the WDK if there were any errors in any of the component services.

APPENDIX B

API DOCUMENTATION

org.gusdb.wsf.service

Class WsfService

```
java.lang.Object
  └org.gusdb.wsf.service.WsfService
```

```
public class WsfService
extends java.lang.Object
```

The WSF Web service entry point.

Constructor Summary

WsfService()

Method Summary

WsfResponse	**invoke**(java.lang.String pluginClassName, java.lang.String projectId, java.lang.String[] paramValues, java.lang.String[] columns) This method is left for backward compatibility purpose
WsfResult	**invokeEx**(java.lang.String pluginClassName, java.lang.String projectId, java.lang.String[] paramValues, java.lang.String[] columns) Client requests to run a plugin by providing the complete class name of the plugin, and the service will invoke the plugin and return the result to the client in tabular format.
java.lang.String	**requestResult**(java.lang.String requestId, int packetId)

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

WsfService

```
public WsfService()
```

Method Detail

invoke

```
public WsfResponse invoke(java.lang.String pluginClassName,
                          java.lang.String projectId,
                          java.lang.String[] paramValues,
                          java.lang.String[] columns)
                   throws ServiceException
```

This method is left for backward compatibility purpose

Parameters:

`pluginClassName` -

`projectId` -

`paramValues` -

`columns` -

Returns:

Throws:

`ServiceException`

---

invokeEx

```
public WsfResult invokeEx(java.lang.String pluginClassName,
                          java.lang.String projectId,
                          java.lang.String[] paramValues,
                          java.lang.String[] columns)
                   throws ServiceException
```

Client requests to run a plugin by providing the complete class name of the plugin, and the service will invoke the plugin and return the result to the client in tabular format.

Parameters:

`pluginClassName` -

`projectId` - The id of the project that invokes the service

`paramValues` - an array of "param=value" pairs. The param and value and separated by the first "="

`cols` -

Returns:

Throws:

`WsfServiceException`

`ServiceException`

---

requestResult

```
public java.lang.String requestResult(java.lang.String requestId,
                                       int packetId)
                                throws ServiceException
```

Throws:

ServiceException

Overview Package Class Tree Deprecated Index Help

PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

---

org.gusdb.wsf.client

Class WsfResponse

```
java.lang.Object
  └ org.gusdb.wsf.client.WsfResponse
```

All Implemented Interfaces:

java.io.Serializable

```
public class WsfResponse
extends java.lang.Object
implements java.io.Serializable
```

See Also:

Serialized Form

Constructor Summary

`WsfResponse()`
`WsfResponse(java.lang.String message, java.lang.String[][] results)`

Method Summary

boolean	`equals(java.lang.Object obj)`
static org.apache.axis.encoding.Deserializer	`getDeserializer(java.lang.String mechType, java.lang.Class _javaType, javax.xml.namespace.QName _xmlType)` Get Custom Deserializer
java.lang.String	`getMessage()` Gets the message value for this WsfResponse.
java.lang.String[][]	`getResults()` Gets the results value for this WsfResponse.
static org.apache.axis.encoding.Serializer	`getSerializer(java.lang.String mechType, java.lang.Class _javaType, javax.xml.namespace.QName _xmlType)` Get Custom Serializer
static org.apache.axis.description.TypeDesc	`getTypeDesc()` Return type metadata object
int	`hashCode()`
void	`setMessage(java.lang.String message)` Sets the message value for this WsfResponse.
void	`setResults(java.lang.String[][] results)` Sets the results value for this WsfResponse.

Methods inherited from class java.lang.Object
clone, finalize, getClass, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

WsfResponse

```
public WsfResponse()
```

WsfResponse

```
public WsfResponse(java.lang.String message,
                   java.lang.String[][] results)
```

Method Detail

getMessage

```
public java.lang.String getMessage()
```

Gets the message value for this WsfResponse.

Returns:
message

---

setMessage

```
public void setMessage(java.lang.String message)
```

Sets the message value for this WsfResponse.

Parameters:
`message` -

---

getResults

```
public java.lang.String[][] getResults()
```

Gets the results value for this WsfResponse.

Returns:
results

---

setResults

```
public void setResults(java.lang.String[][] results)
```

Sets the results value for this WsfResponse.

Parameters:
`results` -

---

equals

```
public boolean equals(java.lang.Object obj)
```

Overrides:

equals in class java.lang.Object

---

hashCode

```
public int hashCode()
```

Overrides:

hashCode in class java.lang.Object

---

getTypeDesc

```
public static org.apache.axis.description.TypeDesc getTypeDesc()
```

Return type metadata object

---

getSerializer

```
public static org.apache.axis.encoding.Serializer getSerializer(java.lang.String mechType,
                                                                 java.lang.Class _javaType,
                                                                 javax.xml.namespace.QName _xmlType)
```

Get Custom Serializer

---

getDeserializer

```
public static org.apache.axis.encoding.Deserializer getDeserializer(java.lang.String mechType,
                                                                     java.lang.Class _javaType,
                                                                     javax.xml.namespace.QName _xmlType)
```

Get Custom Deserializer

---

org.gusdb.wsf.plugin

Class WsfResult

```
java.lang.Object
  └ org.gusdb.wsf.plugin.WsfResult
```

All Implemented Interfaces:

java.io.Serializable

```
public class WsfResult
extends java.lang.Object
implements java.io.Serializable
```

See Also:

Serialized Form

Constructor Summary

`WsfResult()`
`WsfResult(int currentPacket, java.lang.String message, java.lang.String requestId, java.lang.String[][] result, int signal, int totalPackets)`

Method Summary

boolean	`equals(java.lang.Object obj)`
int	`getCurrentPacket()` Gets the currentPacket value for this WsfResult.
static org.apache.axis.encoding.Deserializer	`getDeserializer(java.lang.String mechType, java.lang.Class _javaType, javax.xml.namespace.QName _xmlType)` Get Custom Deserializer
java.lang.String	`getMessage()` Gets the message value for this WsfResult.
java.lang.String	`getRequestId()` Gets the requestId value for this WsfResult.
java.lang.String[][]	`getResult()` Gets the result value for this WsfResult.
static org.apache.axis.encoding.Serializer	`getSerializer(java.lang.String mechType, java.lang.Class _javaType, javax.xml.namespace.QName _xmlType)` Get Custom Serializer
int	`getSignal()` Gets the signal value for this WsfResult.
int	`getTotalPackets()` Gets the totalPackets value for this WsfResult.

static org.apache.axis.description.TypeDesc	**getTypeDesc**() Return type metadata object
int	**hashCode**()
void	**setCurrentPacket**(int currentPacket) Sets the currentPacket value for this WsfResult.
void	**setMessage**(java.lang.String message) Sets the message value for this WsfResult.
void	**setRequestId**(java.lang.String requestId) Sets the requestId value for this WsfResult.
void	**setResult**(java.lang.String[][] result) Sets the result value for this WsfResult.
void	**setSignal**(int signal) Sets the signal value for this WsfResult.
void	**setTotalPackets**(int totalPackets) Sets the totalPackets value for this WsfResult.

Methods inherited from class java.lang.Object
clone, finalize, getClass, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

WsfResult

```
public WsfResult()
```

WsfResult

```
public WsfResult(int currentPacket,
                 java.lang.String message,
                 java.lang.String requestId,
                 java.lang.String[][] result,
                 int signal,
                 int totalPackets)
```

Method Detail

getCurrentPacket

```
public int getCurrentPacket()
```

Gets the currentPacket value for this WsfResult.

Returns:
currentPacket

---

setCurrentPacket

```
public void setCurrentPacket(int currentPacket)
```

Sets the currentPacket value for this WsfResult.

Parameters:
currentPacket -

---

getMessage

```
public java.lang.String getMessage()
```

Gets the message value for this WsfResult.

Returns:
message

---

setMessage

```
public void setMessage(java.lang.String message)
```

Sets the message value for this WsfResult.

Parameters:
message -

---

getRequestId

```
public java.lang.String getRequestId()
```

Gets the requestId value for this WsfResult.

Returns:
requestId

---

setRequestId

```
public void setRequestId(java.lang.String requestId)
```

Sets the requestId value for this WsfResult.

Parameters:
requestId -

---

getResult

```
public java.lang.String[][] getResult()
```

Gets the result value for this WsfResult.

Returns:
result

---

setResult

```
public void setResult(java.lang.String[][] result)
```

Sets the result value for this WsfResult.

Parameters:
`result` -

---

getSignal

```
public int getSignal()
```

Gets the signal value for this WsfResult.

Returns:
signal

---

setSignal

```
public void setSignal(int signal)
```

Sets the signal value for this WsfResult.

Parameters:
`signal` -

---

getTotalPackets

```
public int getTotalPackets()
```

Gets the totalPackets value for this WsfResult.

Returns:
totalPackets

---

setTotalPackets

```
public void setTotalPackets(int totalPackets)
```

Sets the totalPackets value for this WsfResult.

Parameters:
`totalPackets` -

---

equals

```
public boolean equals(java.lang.Object obj)
```

Overrides:
`equals` in class `java.lang.Object`

---

hashCode

```
public int hashCode()
```

Overrides:
`hashCode` in class `java.lang.Object`

---

getTypeDesc

```
public static org.apache.axis.description.TypeDesc getTypeDesc()
```

Return type metadata object

---

getSerializer

```
public static org.apache.axis.encoding.Serializer getSerializer(java.lang.String mechType,
                                                                 java.lang.Class _javaType,
                                                                 javax.xml.namespace.QName _xmlType)
```

Get Custom Serializer

---

getDeserializer

```
public static org.apache.axis.encoding.Deserializer getDeserializer(java.lang.String mechType,
                                                                     java.lang.Class _javaType,
                                                                     javax.xml.namespace.QName _xmlType)
```

Get Custom Deserializer

---

org.gusdb.wsf.plugin

Class WsfServiceException

```
java.lang.Object
  └ java.lang.Throwable
      └ java.lang.Exception
          └ org.gusdb.wsf.plugin.WsfServiceException
```

All Implemented Interfaces:

java.io.Serializable

```
public class WsfServiceException
extends java.lang.Exception
implements java.io.Serializable
```

See Also:

Serialized Form

Constructor Summary

WsfServiceException()
WsfServiceException(java.lang.String message)
WsfServiceException(java.lang.String message, java.lang.Throwable cause)
WsfServiceException(java.lang.Throwable cause)

Method Summary

Methods inherited from class java.lang.Throwable
fillInStackTrace, getCause, getLocalizedMessage, getMessage, getStackTrace, initCause, printStackTrace, printStackTrace, printStackTrace, setStackTrace, toString

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructor Detail

WsfServiceException

```
public WsfServiceException()
```

WsfServiceException

```
public WsfServiceException(java.lang.String message)
```

Parameters:

message -

WsfServiceException

```
public WsfServiceException(java.lang.String message,
                           java.lang.Throwable cause)
```

Parameters:

message -

cause -

WsfServiceException

```
public WsfServiceException(java.lang.Throwable cause)
```

Parameters:

cause -

---

org.gusdb.wsf.service

Class WsfResponse

```
java.lang.Object
  └org.gusdb.wsf.service.WsfResponse
```

```
public class WsfResponse
extends java.lang.Object
```

Constructor Summary

WsfResponse()

Method Summary

java.lang.String	**getMessage**()
java.lang.String[][]	**getResults**()
void	**setMessage**(java.lang.String message)
void	**setResults**(java.lang.String[][] results)

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

WsfResponse

```
public WsfResponse()
```

Method Detail

getMessage

```
public java.lang.String getMessage()
```

Returns:

Returns the message.

---

setMessage

```
public void setMessage(java.lang.String message)
```

Parameters:

`message` - The message to set.

---

getResults

```
public java.lang.String[][] getResults()
```

Returns:

Returns the results.

---

setResults

```
public void setResults(java.lang.String[][] results)
```

Parameters:

`results` - The results to set.

---

org.gusdb.wsf.client

Class WsfServiceServiceLocator

```
java.lang.Object
  └org.apache.axis.client.Service
      └org.gusdb.wsf.client.WsfServiceServiceLocator
```

All Implemented Interfaces:

WsfServiceService

```
public class WsfServiceServiceLocator
extends org.apache.axis.client.Service
implements WsfServiceService
```

Constructor Summary

WsfServiceServiceLocator()
WsfServiceServiceLocator(org.apache.axis.EngineConfiguration config)
WsfServiceServiceLocator(java.lang.String wsdlLoc, javax.xml.namespace.QName sName)

Method Summary

java.rmi.Remote	**getPort**(java.lang.Class serviceEndpointInterface) For the given interface, get the stub implementation.
java.rmi.Remote	**getPort**(javax.xml.namespace.QName portName, java.lang.Class serviceEndpointInterface) For the given interface, get the stub implementation.
java.util.Iterator	**getPorts**()
javax.xml.namespace.QName	**getServiceName**()
WsfService	**getWsfService**()
WsfService	**getWsfService**(java.net.URL portAddress)

java.lang.String	**getWsfServiceAddress**()
java.lang.String	**getWsfServiceWSDDServiceName**()
void	**setEndpointAddress**(javax.xml.namespace.QName portName, java.lang.String address) Set the endpoint address for the specified port name.
void	**setEndpointAddress**(java.lang.String portName, java.lang.String address) Set the endpoint address for the specified port name.
void	**setWsfServiceEndpointAddress**(java.lang.String address)
void	**setWsfServiceWSDDServiceName**(java.lang.String name)

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

WsfServiceServiceLocator

```
public WsfServiceServiceLocator()
```

WsfServiceServiceLocator

```
public WsfServiceServiceLocator(org.apache.axis.EngineConfiguration config)
```

WsfServiceServiceLocator

```
public WsfServiceServiceLocator(java.lang.String wsdlLoc,
                                javax.xml.namespace.QName sName)
                         throws javax.xml.rpc.ServiceException
```

Throws:

javax.xml.rpc.ServiceException

Method Detail

getWsfServiceAddress

```
public java.lang.String getWsfServiceAddress()
```

Specified by:

getWsfServiceAddress in interface WsfServiceService

---

getWsfServiceWSDDServiceName

```
public java.lang.String getWsfServiceWSDDServiceName()
```

---

setWsfServiceWSDDServiceName

```
public void setWsfServiceWSDDServiceName(java.lang.String name)
```

---

getWsfService

```
public WsfService getWsfService()
                        throws javax.xml.rpc.ServiceException
```

Specified by:

getWsfService in interface WsfServiceService

Throws:

javax.xml.rpc.ServiceException

---

getWsfService

```
public WsfService getWsfService(java.net.URL portAddress)
                        throws javax.xml.rpc.ServiceException
```

Specified by:

getWsfService in interface WsfServiceService

Throws:

javax.xml.rpc.ServiceException

---

setWsfServiceEndpointAddress

```
public void setWsfServiceEndpointAddress(java.lang.String address)
```

---

getPort

```
public java.rmi.Remote getPort(java.lang.Class serviceEndpointInterface)
                        throws javax.xml.rpc.ServiceException
```

For the given interface, get the stub implementation. If this service has no port for the given

interface, then ServiceException is thrown.

Throws:

`javax.xml.rpc.ServiceException`

---

getPort

```
public java.rmi.Remote getPort(javax.xml.namespace.QName portName,
                               java.lang.Class serviceEndpointInterface)
                        throws javax.xml.rpc.ServiceException
```

For the given interface, get the stub implementation. If this service has no port for the given interface, then ServiceException is thrown.

Throws:

`javax.xml.rpc.ServiceException`

---

getServiceName

```
public javax.xml.namespace.QName getServiceName()
```

---

getPorts

```
public java.util.Iterator getPorts()
```

---

setEndpointAddress

```
public void setEndpointAddress(java.lang.String portName,
                               java.lang.String address)
                        throws javax.xml.rpc.ServiceException
```

Set the endpoint address for the specified port name.

Throws:

`javax.xml.rpc.ServiceException`

---

setEndpointAddress

```
public void setEndpointAddress(javax.xml.namespace.QName portName,
                               java.lang.String address)
                        throws javax.xml.rpc.ServiceException
```

Set the endpoint address for the specified port name.

Throws:

`javax.xml.rpc.ServiceException`

Overview Package Class Tree Deprecated Index Help

PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

---

org.gusdb.wsf.client

Class WsfServiceSoapBindingStub

```
java.lang.Object
  └org.apache.axis.client.Stub
      └org.gusdb.wsf.client.WsfServiceSoapBindingStub
```

All Implemented Interfaces:

java.rmi.Remote, WsfService

```
public class WsfServiceSoapBindingStub
extends org.apache.axis.client.Stub
implements WsfService
```

Constructor Summary

WsfServiceSoapBindingStub()
WsfServiceSoapBindingStub(javax.xml.rpc.Service service)
WsfServiceSoapBindingStub(java.net.URL endpointURL, javax.xml.rpc.Service service)

Method Summary

protected org.apache.axis.client.Call	**createCall**()
WsfResponse	**invoke**(java.lang.String pluginClassName, java.lang.String projectId, java.lang.String[] paramValues, java.lang.String[] columns)
WsfResult	**invokeEx**(java.lang.String pluginClassName, java.lang.String projectId, java.lang.String[] paramValues, java.lang.String[] columns)
java.lang.String	**requestResult**(java.lang.String requestId, int packetId)

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

WsfServiceSoapBindingStub

```
public WsfServiceSoapBindingStub()
                           throws org.apache.axis.AxisFault
```

Throws:

org.apache.axis.AxisFault

WsfServiceSoapBindingStub

```
public WsfServiceSoapBindingStub(java.net.URL endpointURL,
                                 javax.xml.rpc.Service service)
                           throws org.apache.axis.AxisFault
```

Throws:

org.apache.axis.AxisFault

WsfServiceSoapBindingStub

```
public WsfServiceSoapBindingStub(javax.xml.rpc.Service service)
                           throws org.apache.axis.AxisFault
```

Throws:

org.apache.axis.AxisFault

Method Detail

createCall

```
protected org.apache.axis.client.Call createCall()
                                          throws java.rmi.RemoteException
```

Throws:

java.rmi.RemoteException

---

invoke

```
public WsfResponse invoke(java.lang.String pluginClassName,
                          java.lang.String projectId,
                          java.lang.String[] paramValues,
                          java.lang.String[] columns)
                   throws java.rmi.RemoteException
```

Specified by:

invoke in interface WsfService

Throws:

java.rmi.RemoteException

---

invokeEx

```
public WsfResult invokeEx(java.lang.String pluginClassName,
                          java.lang.String projectId,
                          java.lang.String[] paramValues,
                          java.lang.String[] columns)
                   throws java.rmi.RemoteException
```

Specified by:

invokeEx in interface WsfService

Throws:

java.rmi.RemoteException

---

requestResult

```
public java.lang.String requestResult(java.lang.String requestId,
                                      int packetId)
                               throws java.rmi.RemoteException
```

Specified by:

requestResult in interface WsfService

Throws:

java.rmi.RemoteException

---

org.gusdb.wsf.plugin

Class WsfPlugin

```
java.lang.Object
  └ org.gusdb.wsf.plugin.WsfPlugin
```

All Implemented Interfaces:

IWsfPlugin

```
public abstract class WsfPlugin
extends java.lang.Object
implements IWsfPlugin
```

The WsfPlugin provides the common routines a plugin needs to simplify the development of new WSF plugins.

Field Summary

protected Logger	**logger** The logger for this plugin.
protected static java.lang.String	**newline**
protected ServletContext	**servletContext**

Constructor Summary

WsfPlugin() Initialize a plugin with empty properties
WsfPlugin(java.lang.String propertyFile) Initialize a plugin and assign a property file to it

Method Summary

protected abstract WsfResult	**execute**(java.lang.String queryName, java.util.Map<java.lang.String,java.lang.String> params, java.lang.String[] orderedColumns) The plugin should implement this method to do the real job, for example, to invoke an application, and prepare the results into tabular format, and then return the results.
protected abstract java.lang.String[]	**getColumns**() The Plugin needs to provides a list of the columns expected in the result; the base class will use this template method in the input validation process.
protected java.lang.String	**getProperty**(java.lang.String propertyName)
protected abstract java.lang.String[]	**getRequiredParameterNames**() The Plugin needs to provide a list of required parameter names; the base class will use this template method in the input validation process.

WsfResult	**invoke**(java.lang.String projectId, java.util.Map<java.lang.String,java.lang.String> params, java.lang.String[] orderedColumns) The service will call the plugin through this interface.
protected int	**invokeCommand**(java.lang.String[] command, java.lang.StringBuffer result, long timeout)
void	**setLogger**(Logger logger)
protected void	**validateColumns**(java.lang.String[] orderedColumns)
protected abstract void	**validateParameters**(java.util.Map<java.lang.String,java.lang.String> params)

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

newline

```
protected static final java.lang.String newline
```

---

logger

```
protected Logger logger
```

The logger for this plugin. It is a recommended way to record standard output and error messages.

---

servletContext

```
protected ServletContext servletContext
```

Constructor Detail

WsfPlugin

```
public WsfPlugin()
```

Initialize a plugin with empty properties

WsfPlugin

```
public WsfPlugin(java.lang.String propertyFile)
          throws WsfServiceException
```

Initialize a plugin and assign a property file to it

Parameters:

`propertyFile` - the name of the property file. The base class will resolve the path to this file, which should be under the WEB-INF of axis' webapps.

Throws:

`WsfServiceException`

Method Detail

getRequiredParameterNames

```
protected abstract java.lang.String[] getRequiredParameterNames()
```

The Plugin needs to provide a list of required parameter names; the base class will use this template method in the input validation process.

Returns:

returns an array the names of the required parameters

---

getColumns

```
protected abstract java.lang.String[] getColumns()
```

The Plugin needs to provides a list of the columns expected in the result; the base class will use this template method in the input validation process.

Returns:

returns an array the columns expected in the result

---

execute

```
protected abstract WsfResult execute(java.lang.String queryName,
                                     java.util.Map<java.lang.String,java.lang.String> params,
                                     java.lang.String[] orderedColumns)
                              throws WsfServiceException
```

The plugin should implement this method to do the real job, for example, to invoke an application, and prepare the results into tabular format, and then return the results.

Parameters:

`queryName` - the name of the query that invokes this plugin

`params` - The Map of parameters given by the client

`orderedColumns` - The ordered columns assigned by the client. each of the columns must match with one column sepecified in getColumns() by the plugin. The The plugin is responsible to re-format the result following the order of the columns defined here.

Returns:

returns the result in 2-dimensional array of strings format.

Throws:

`WsfServiceException`

---

setLogger

```
public void setLogger(Logger logger)
```

Specified by:

setLogger in interface IWsfPlugin

Parameters:

`logger` - set a different logger to the plugin. The WsfService will assign a specific logger to each plugin.

---

invoke

```
public final WsfResult invoke(java.lang.String projectId,
                              java.util.Map<java.lang.String,java.lang.String> params,
                              java.lang.String[] orderedColumns)
                       throws WsfServiceException
```

The service will call the plugin through this interface. Plugin cannot override this method from the base class, instead it should implement execute() method.

Specified by:

invoke in interface IWsfPlugin

Returns:

Throws:

WsfServiceException

See Also:

`org.gusdb.wsf.IWsfPlugin#invoke(java.util.Map, java.lang.String[])`

---

validateParameters

```
protected abstract void validateParameters(java.util.Map<java.lang.String,java.lang.String> params)
                                    throws WsfServiceException
```

Throws:

WsfServiceException

---

validateColumns

```
protected void validateColumns(java.lang.String[] orderedColumns)
                        throws WsfServiceException
```

Throws:

WsfServiceException

---

getProperty

```
protected java.lang.String getProperty(java.lang.String propertyName)
```

---

invokeCommand

```
protected int invokeCommand(java.lang.String[] command,
                            java.lang.StringBuffer result,
                            long timeout)
                     throws java.io.IOException
```

Parameters:

`command` - the command array. If you have param values with spaces in it, put the value into one cell to

avoid the value to be splitted.
`timeout` - the maximum allowed time for the command to run, in seconds
`result` - Contains raw output of the command.

Returns:

the exit code of the invoked command

Throws:

`java.io.IOException`

Overview Package Class Tree Deprecated Index Help

PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

---

org.apidb.apicomplexa.wsfplugin.apifed

Class ApiFedPlugin

```
java.lang.Object
  └WsfPlugin
      └org.apidb.apicomplexa.wsfplugin.apifed.ApiFedPlugin
```

```
public class ApiFedPlugin
extends WsfPlugin
```

Field Summary

static java.lang.String	**COLUMN_RETURN**
static java.lang.String	**MAPPING_FILE**
static java.lang.String	**PARAM_COLUMNS**
static java.lang.String	**PARAM_ORGANISMS**
static java.lang.String	**PARAM_PARAMETERS**
static java.lang.String	**PARAM_PROCESSNAME**
static java.lang.String	**PARAM_QUERY**
static java.lang.String	**PARAM_SET_NAME**
static java.lang.String	**PROPERTY_FILE**
static java.lang.String	**VERSION**

Constructor Summary

ApiFedPlugin()
Constructor for ApiFedPluginClass This class acts as the federation service for the EuPathDB Web service Federation system

Method Summary

protected WsfResult	**execute**(java.lang.String queryName, java.util.Map<java.lang.String,java.lang.String> params, java.lang.String[] orderedColumns) Main function in this class to perform the function of federation service Decomp query based on organism and distributes queries to component sites as needed
protected java.lang.String[]	**getColumns**()
protected java.lang.String[]	**getRequiredParameterNames**()
protected void	**validateColumns**(java.lang.String[] orderedColumns)
protected void	**validateParameters**(java.util.Map<java.lang.String,java.lang.String> par

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

PROPERTY_FILE

public static final java.lang.String **PROPERTY_FILE**

See Also:
Constant Field Values

MAPPING_FILE

public static final java.lang.String **MAPPING_FILE**

See Also:
Constant Field Values

VERSION

public static final java.lang.String **VERSION**

See Also:
Constant Field Values

PARAM_SET_NAME

public static final java.lang.String **PARAM_SET_NAME**

See Also:
Constant Field Values

PARAM_PROCESSNAME

```
public static final java.lang.String PARAM_PROCESSNAME
```

See Also:
Constant Field Values

PARAM_PARAMETERS

```
public static final java.lang.String PARAM_PARAMETERS
```

See Also:
Constant Field Values

PARAM_COLUMNS

```
public static final java.lang.String PARAM_COLUMNS
```

See Also:
Constant Field Values

PARAM_ORGANISMS

```
public static final java.lang.String PARAM_ORGANISMS
```

See Also:
Constant Field Values

PARAM_QUERY

```
public static final java.lang.String PARAM_QUERY
```

See Also:
Constant Field Values

COLUMN_RETURN

```
public static final java.lang.String COLUMN_RETURN
```

See Also:
Constant Field Values

Constructor Detail

ApiFedPlugin

```
public ApiFedPlugin()
            throws WsfServiceException
```

Constructor for ApiFedPluginClass This class acts as the federation service for the EuPathDB Web service Federation system

Throws:

`WsfServiceException`

Method Detail

getRequiredParameterNames

```
protected java.lang.String[] getRequiredParameterNames()
```

---

getColumns

```
protected java.lang.String[] getColumns()
```

---

validateParameters

```
protected void validateParameters(java.util.Map<java.lang.String,java.lang.String> params)
                          throws WsfServiceException
```

Throws:

`WsfServiceException`

---

validateColumns

```
protected void validateColumns(java.lang.String[] orderedColumns)
```

---

execute

```
protected WsfResult execute(java.lang.String queryName,
                            java.util.Map<java.lang.String,java.lang.String> params,
                            java.lang.String[] orderedColumns)
                     throws WsfServiceException
```

Main function in this class to perform the function of federation service Decomposes query based on organism and distributes queries to component sites as needed

Parameters:

`queryName` - name of the query to be executed on components
`params` - Map of parameter name and parameter values
`orderedColumns` - Array of column name to be returned

Returns:

WsfResult containing combined results of component services

Throws:

`WsfServiceException`

See Also:

`org.gusdb.wsf.WsfPlugin#execute(java.util.Map, java.lang.String[])`

Overview Package Class Tree Deprecated Index Help

PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

---

org.apidb.apicomplexa.wsfplugin.wdkquery

Class WdkQueryPlugin

```
java.lang.Object
  └WsfPlugin
      └org.apidb.apicomplexa.wsfplugin.wdkquery.WdkQueryPlugin
```

```
public class WdkQueryPlugin
extends WsfPlugin
```

Field Summary

static java.lang.String	**COLUMN_RETURN**
static java.lang.String	**GUS_HOME**
static java.lang.String	**MODEL_NAME**
static java.lang.String	**PARAM_COLUMNS**
static java.lang.String	**PARAM_PARAMETERS**
static java.lang.String	**PROPERTY_FILE**
static java.lang.String	**SITE_MODEL**
static java.lang.String	**VERSION**

Constructor Summary

WdkQueryPlugin()
Constructor for the WdkQueryPlugin object This object access WDK to execute the provided query acts as the component service for the EuPathDB federation system

Method Summary

protected java.lang.String	**convertDatasetId2DatasetChecksum**(java.lang.St handles the dataset parameters from WDK must values to pass the correct information

protected java.util.Map<java.lang.String,java.lang.String>	**convertParams**(User user, java.util.Map<java.lang.String,java.lang.Stri converts parameters from internal to external va
protected WsfResult	**execute**(java.lang.String invokeKey, java.util.Map<java.lang.String,java.lang.Stri java.lang.String[] orderedColumns) access WDK and executes the proper query with given set of columns
protected java.lang.String[]	**getColumns**()
protected java.lang.String[]	**getRequiredParameterNames**()
protected java.lang.String[][]	**handleEnumParameters**(Param p, java.lang.Strir Allows the federation to work for all types of pa enumParameters in the portal Model
protected java.lang.String[][]	**results2StringArray**(Column[] cols, ResultList Converts results into a String[][] to be returned t
protected void	**validateColumns**(java.lang.String[] orderedCol
protected void	**validateParameters**(java.util.Map<java.lang.St
protected boolean	**validateSingleValues**(AbstractEnumParam p, jav validated the inputs to ensure only allowed valu

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

PROPERTY_FILE

```
public static final java.lang.String PROPERTY_FILE
```

See Also:

Constant Field Values

MODEL_NAME

```
public static final java.lang.String MODEL_NAME
```

See Also:

Constant Field Values

GUS_HOME

```
public static final java.lang.String GUS_HOME
```

See Also:
Constant Field Values

VERSION

```
public static final java.lang.String VERSION
```

See Also:
Constant Field Values

PARAM_PARAMETERS

```
public static final java.lang.String PARAM_PARAMETERS
```

See Also:
Constant Field Values

PARAM_COLUMNS

```
public static final java.lang.String PARAM_COLUMNS
```

See Also:
Constant Field Values

SITE_MODEL

```
public static final java.lang.String SITE_MODEL
```

See Also:
Constant Field Values

COLUMN_RETURN

```
public static final java.lang.String COLUMN_RETURN
```

See Also:
Constant Field Values

Constructor Detail

WdkQueryPlugin

```
public WdkQueryPlugin()
               throws WsfServiceException
```

Constructor for the WdkQueryPlugin object This object access WDK to execute the provided query acts as the component service for the EuPathDB federation system

Throws:

`WsfServiceException`

Method Detail

getRequiredParameterNames

```
protected java.lang.String[] getRequiredParameterNames()
```

---

getColumns

```
protected java.lang.String[] getColumns()
```

---

validateParameters

```
protected void validateParameters(java.util.Map<java.lang.String,java.lang.String> params)
                           throws WsfServiceException
```

Throws:

`WsfServiceException`

---

validateColumns

```
protected void validateColumns(java.lang.String[] orderedColumns)
```

---

execute

```
protected WsfResult execute(java.lang.String invokeKey,
                            java.util.Map<java.lang.String,java.lang.String> params,
                            java.lang.String[] orderedColumns)
                     throws WsfServiceException
```

access WDK and executes the proper query with the given parameters returns the given set of columns

Parameters:

`invokeKey` - query name that will be executed by the service
`params` - parameters to be given as input to the query
`orderedColumns` - list of columns that should be returned by the service

Returns:

Object containing the results of the query from the WDK

Throws:

`WsfServiceException`

See Also:

`org.gusdb.wsf.WsfPlugin#execute(java.util.Map, java.lang.String[])`

---

convertDatasetId2DatasetChecksum

```
protected java.lang.String convertDatasetId2DatasetChecksum(java.lang.String sig_id)
                                                     throws java.lang.Exception
```

handles the dataset parameters from WDK must convert form internal to external values to pass the correct information

Parameters:

`sid_id` - signature of the user being used for this data set

Returns:

converted dataset id as string

Throws:

`java.lang.Exception`

---

convertParams

```
protected java.util.Map<java.lang.String,java.lang.String> convertParams(User user,
                                                                          java.util.Map<java.
                                                                          Param[] q)
```

converts parameters from internal to external values and validated values

Parameters:

`user` - User object that initiated the query on the portal

`p` - parameters for the query from the portal

`q` - parameters for the query from the WDK Model

Returns:

converted parameters

---

results2StringArray

```
protected java.lang.String[][] results2StringArray(Column[] cols,
                                                   ResultList result)
                                            throws WdkModelException
```

Converts results into a String[][] to be returned to the portal

Parameters:

`cols` - Array of columns to be returned

`result` - ResultList from the WDK containing the results of the query

Returns:

results as a two-demensional array of String

Throws:

`WdkModelException`

---

validateSingleValues

```
protected boolean validateSingleValues(AbstractEnumParam p,
                                       java.lang.String value)
                                throws WdkModelException,
                                       java.security.NoSuchAlgorithmException,
                                       java.sql.SQLException,
                                       JSONException,
                                       WdkUserException
```

validated the inputs to ensure only allowed values

Parameters:

`p` - Parameter from WDK Model

`value` - Value passed in from the portal

Returns:

true if value is valid for the given parameter

Throws:

`WdkModelException`

`java.security.NoSuchAlgorithmException`

`java.sql.SQLException`

`JSONException`

`WdkUserException`

---

handleEnumParameters

```
protected java.lang.String[][] handleEnumParameters(Param p,
                                                    java.lang.String[] ordCols)
                                             throws WdkModelException,
                                                    java.security.NoSuchAlgorithmException,
                                                    java.sql.SQLException,
                                                    JSONException,
                                                    WdkUserException
```

Allows the federation to work for all types of parameters to eliminate the need for enumParameters in the portal Model

Parameters:

`p` - Parameter to get from the model

`ordCols` - values to return for the portal

Returns:

results of the parameter from the this component WDK Model

Throws:

`WdkModelException`

`java.security.NoSuchAlgorithmException`

`java.sql.SQLException`

`JSONException`

`WdkUserException`

---

org.gusdb.wdk.controller.action

Class ProcessRESTAction

```
java.lang.Object
  └ ShowQuestionAction
      └ org.gusdb.wdk.controller.action.ProcessRESTAction
```

```
public class ProcessRESTAction
extends ShowQuestionAction
```

This Action is called by the ActionServlet when a REST service is invoked. The path is used to determine the Question to be asked and the parameters are pulled from the URL The results are returned in a either XML or JSON format.

Constructor Summary

`ProcessRESTAction()`

Method Summary

ActionForward	`execute(ActionMapping mapping, ActionForm form, HttpServletRequest request, HttpServletResponse response)` Function to execute the question.
protected void	`reportError(HttpServletResponse resp, java.util.Map<java.lang.String,java.lang.String> msg, java.lang.String errType, java.lang.String errCode, java.lang.String type)` Handles errors from the execute function and outputs them in the given format (XML or JSON)

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

ProcessRESTAction

```
public ProcessRESTAction()
```

Method Detail

execute

```
public ActionForward execute(ActionMapping mapping,
                             ActionForm form,
                             HttpServletRequest request,
                             HttpServletResponse response)
                      throws java.lang.Exception
```

Function to execute the question.

Parameters:

`mapping` - Mapping for the actions in the servlet
`form` - Form from the question page (not use in this environment)
`request` - HttpServletREquest Object
`response` - HttpServletResponse Object

Returns:

null

Throws:

`java.lang.Exception`

---

reportError

```
protected void reportError(HttpServletResponse resp,
                           java.util.Map<java.lang.String,java.lang.String> msg,
                           java.lang.String errType,
                           java.lang.String errCode,
                           java.lang.String type)
                    throws java.io.IOException
```

Handles errors from the execute function and outputs them in the given format (XML or JSON)

Parameters:

`resp` - HttpServletResponse Object
`msg` - Map of error messages included in an exception
`errType` - Name this type of error
`errCode` - Code for the error
`type` - type of output ('xml' or 'json')

Throws:

`java.io.IOException`
