To appear at the Joint Workshop on Web Services Modeling and Implementation Using Sound Web Engineering Practices and Methods, Architectures & Technologies for e-Service Engineering (SMIWEB-MATeS 2006) workshop held in conjunction with the Sixth International Conference on Web Engineering (ICWE'06). Palo Alto, California, July 2006.

Integrating Bioinformatic Data Sources over the SFSU ER Design Tools XML Databus

Yan Liu, M.S.
Computer Science Department
San Francisco State University
1600 Holloway Avenue
San Francisco, CA 94132 USA
1-415-338-1008
yliu0902@sfsu.edu

Sorna Vincent, M.S.
Computer Science Department
San Francisco State University
1600 Holloway Avenue
San Francisco, CA 94132 USA
1-415-338-1008
vmahima@yahoo.com

Marguerite C. Murphy, PhD
Computer Science Department
San Francisco State University
1600 Holloway Avenue
San Francisco, CA 94132 USA
1-415-338-2261
mmurphy@sfsu.edu

ABSTRACT
The SFSU ER Design Tools were developed to support database design and data integration over multiple implementation data models. These tools allow users to enter and view Entity Relationship (ER) schemas and to translate ER schemas into a variety of equivalent implementation schemas, including Relational (ANSI SQL2), Object Oriented (ODMG 3.0), Spreadsheet (Universal Relation with associated functional dependencies) and W3C XML DTD. In addition, for each implementation data model, the Tools generate DDL statements to *create* a database, as well as simple JDBC/ODBC based code to *dump* stored data into an XML file and to *load* data from an XML file into a database. Data can be transferred from one data store to another over an HTTP based XML Databus. In this paper we describe the design and implementation of our XML Databus using Web Services, as well as a new strategy to support integration of bioinformatics data sets. We first manually identify semantically equivalent attributes in both schemas, then automatically join the corresponding data sets into a single integrated collection of XML formatted data. Our software is operational, and preliminary performance measurements over DTD and data downloaded from the NIH-NCBI Web site show that our strategy is feasible for moderately sized data sets.

Categories and Subject Descriptors
H.2.1 [**Database Management**]: Logical Design – *data models, schemas and subschemas.*

General Terms
Algorithms, Measurement, Design, Standardization

Keywords
Web Services data integration, join processing, schema integration, distributed query processing.

1. INTRODUCTION
Various high throughput biological experimental techniques have generated and continue to generate large amounts of diverse biological data, such as gene expression and sequence data, phenotype data, protein data, disease characteristics, molecular structures, microarray data, etc. Due to their diverse nature, these data are widely distributed in a variety of remote data stores using different data models (i.e. ANSI SQL2 Relational, Native XML, Spreadsheet, Native XML and ODMG3.0) and are frequently made available over the Internet. Biological data stored in these data stores can be complementary or overlapping, which means two data stores may store different parts of the schema, and two data stores may have

their schemas overlapped. However, Biologists often need an integrated or combined view of different data collections from these data stores for their research. In the most simplistic scenario, interesting collections of data from multiple data stores are syntactically converted to a single data representation to allow processing over the joined or combined (union) data sets. With the increasing numbers of online biological data collections available, fully manual integration has become practically infeasible.

Although integration techniques have been studied in the biological community for many years, these strategies are still in the exploratory phase due to the complexity of biological data and data stores. Currently, there are two broad types of integration techniques being used to integrate distributed heterogeneous biological data and data stores: link driven integration and view integration [5]. In the link-driven integration technique, users start by extracting entries of interest from one data store and then jump to other related data stores by following Web links embedded in the data stores. On the other hand, in view integration, the schemas of the underlying data collections are merged to form a global schema, and then users query this global schema using a high-level query language (e.g. SQL, QQL, or XQuery). View integration can be further divided into two subtypes, data warehousing and data federation, depending on whether the global schema is instantiated or not. Data warehousing integration consists of materializing the data from multiple databases into a local database, called the data warehouse, and executing all queries on the data contained in the warehouse rather than over the original databases. In data federation integration, a global schema is formed by merging the schemas of the underlying databases using some common model

(e.g. relational, complex value, or object-oriented model). Users query this global schema using a high-level query language. The system then determines what portion of the global query can be answered by which data sources, ships local queries off to the appropriate databases, and combines the answers from the various databases to produce an answer to the global query. As XML is the Internet standard for data representation and exchange, and Web Services provide seamless interoperation of resources using well-defined standards (SOAP/XML, WSDL, and UDDI), these techniques are widely applied in designing integration techniques.

The SFSU ER Design Tools [14] can generate schemas for different data models (relational, XML, ODMG, and spreadsheet) to help users design a database. These tools also provide functionality to exchange data among various data models. Currently, these tools are constructed using a client-server architecture and the server side is re-implemented as a Web Service [2]. In addition, in order to resolve the practical data integration problems associated with transferring data among heterogeneous biological data stores using the ER Design Tools, a prototype for heterogeneous Data Transfer has been implemented using Web Services [1].

This project assumes the interrelations between heterogeneous biological data stores are "complementary", which means the global schema is formed by merging the schemas of the individual data stores, and queries across these data stores need to use a relational join operation to combine data. By incorporating the Data Transfer Web Service along with the ER Design Tool Web Service, a new Web Service is built. This Web Service executes distributed joins among distributed heterogeneous data store Web Services by first transferring data from different remote sites to a common site, then asking users to identify attributes (tags) that are synonyms, and from this information forming a global schema. Next, data are loaded into the local relational data store to be combined by executing a join query. The join result is dumped into an XML formatted file.

The project presented in this paper is implemented using Java Web Services Developer Pack version 1.4 (JWSDP 1.4). The JWSDP package includes Java API for XML processing (JAXP) v1.0.3, and Java API for XML-based RPC (JAX-RPC) v1.1.2. JAXP is used to process XML documents, and JAX-RPC is used to create remote procedure call based Web Services, along with SOAP for a communication framework and HTTP for the transport protocol over the Internet.

To do performance experiments, this project uses a large collection of data sets downloaded from the NIH-NCBI Web site. The National Center for Biotechnology Information (NCBI) is part of the National Institutes of Health (NIH) and is a major public provider responsible for massive amounts of biological data. These XML formatted data sets were converted from the NIH-NCBI representation to the XML format used by the ER Design Tools [6], then loaded into a relational data store and a spreadsheet for join execution.

This paper is a condensed version of [12]. It describes the design and implementation of a Web Services join using the SFSU ER Design Tools XML Databus. It consists of six sections and is organized as follows. First, section two discusses background information about the SFSU ER Design Tool and its extensions for working with biological data transformation and transfer. Next, section three provides a discussion of the major design decisions. Section four describes the user integration interface and section five presents our preliminary experimental performance results. We conclude and summarize our contributions in section six.

2. BACKGROUND

The SFSU ER Design Tools were developed by Computer Science graduate students under the guidance of Professor Marguerite C. Murphy at San Francisco State University. These tools are used for data modeling, which is the first step for designing a database. These tools allow users to enter an ER schema (or a UML schema) to describe data structures in an application using entities, attributes, relationships between entities, and integrity constraints. Then the tools can map this high level ER schema to an implementation data schema (e.g. relational, XML, ODMG, or spreadsheet) and generate both an English Description and a Constraints Description for the ER schema. In addition, the ER Tools provide a 3D view of the ER schema.

The ER Design Tools were first developed as a Java applet. With the functionalities provided by the tool increasing, the tool was then re-implemented as a Java client-server application. In order to make the tool more widely available, the server side was re-implemented again by Sushma Prasad [2] using Web Services so that anyone can discover this Web Service by searching in a Service registry and writing a client application in C++, Java or other languages to communicate with the ER Design Tool server written in Java.

Sorna Vincent [1] implemented the extension of the ER Design Tools for data transfer between different heterogeneous biological data stores using Web Services. A prototype system was built to integrate heterogeneous data stores. Data is represented using a global ER Schema common to all data stores, which is automatically mapped into each of the representation data models. Each data store is wrapped into a Data Transfer Web Service. The client application uses the Data Transfer Web Service to ask remote data stores to dump data from the native database format to an application specific XML format, transfer the data to a destination data store, and then load the data into the destination data store by translating the XML formatted data into the native data representation.

Tsu-Yao Jen [6] implemented the extensions of the ER Design Tools for biological data transformation. The NIH-NCBI designs database schema and specifications for representation of various forms of molecular biology information, and maintains a large repository of publicly available data sets which can be accessed as XML files. In order to use the ER Design Tools with biological data, the XML formatted data sets downloaded from the NIH-NCBI Web site are converted from the NIH-NCBI representation to the XML format used by the ER Design Tools. This is done using W3C standard Extensible Stylesheet Language Transformations (XSLT). After the NIH-NCBI data sets are transformed, they can go through the ER Design Tool XML Databus and be automatically loaded into different heterogeneous data stores (e.g. relational, spreadsheet, and ODMG).

3. DESIGN OVERVIEW

This section describes the design of this project. In Section 3.1, the goal of this project is described. In Section 3.2, the design decisions for the new join Web Service are explained. In Section 3.3, the overall architecture of the project software is presented. In

Section 3.4, the implementation class diagram is discussed. In Section 3.5, the implementation workflow diagram is described.

3.1 Introduction

The ER Design Tools and its extensions can transfer data between different data stores using different data models (e.g. relational database, spreadsheet, ODMG). Each data store is wrapped into a Web Service interface including the *createDB*(), *dump*() and *load*() operations. Data in a data store can be dumped into XML Databus (ER tool specific XML format[1]) from its native data format, and data in XML Databus format can be loaded into any data store on the Databus. The goal of this work is to design and implement a new join web service that combines data described using different (but overlapping) schemas into a single collection of data. The data collections to be combined initially reside in different ER Tools Web Services data stores.

3.2 Design decisions

This project uses NCBI TinySeq and GBSeq datasets to illustrate the integration process and for performance measurements. Joining biological datasets is challenging due to non-uniformity of the data models being used to represent information (syntactic heterogeneity), non-uniformity in representing data (semantic differences), and the existence of many autonomous Web-based data sources. Web Services are regarded as a good solution for syntactic heterogeneity but not for semantic differences. For example, the TinySeq dataset has an attribute named TSeq_accver, which has a different name in the GBSeq dataset: GBSeq_accession_version. The GBSeq dataset also has an attribute named GBSeq_primary_accession. Furthermore, NCBI only provides DTDs for TSeq and GBSeq datasets. Web Services have no way to tell that TSeq_accver is the same as GBSeq_accession_version but different from GBSeq_primary_accession. In order to make this project more flexible and practical for integrating external biological data stores, users are allowed to manually select common join attributes over the two datasets. A user interface is developed, which allows the user to specify the JDBC attributes needed to connect to both data stores. Using this information, the client connects to the data sources and transfers data to the client. Also using the user interface, the user can browse through the fields stored in both data stores and identify common join attributes as well as the attributes to be displayed in the result. The join attributes are selected in such a way that the paired attributes are semantically identical, but may be named the same or differently in both of the data stores. Once the join attributes and the display attributes have been selected, the user can initiate the join operation.

Based on the assumption that the interrelations between heterogeneous biological data stores are "complementary", in theory a global schema should be formed by merging the schemas of these datasets and used for loading the datasets. The join strategy adopted is to load datasets into a relational database and to use a relational join query to join the datasets. So the global schema in XML format needs to be converted to its counterpart in the relational database; which consists of individual relation schemas and foreign key constraints. But loading data files separately into the relational database using individual schemas (not the global schema) also generates individual relation schemas and foreign key constraints. The global schema in the relational database can be produced by combining these individual relation schemas plus the missing referential constraints involving the join attributes (i.e. the semantically identical attributes). These additional constraints can be derived based on the user-specified join attributes. In order to improve the efficiency of this project, the global schema in XML format is not materialized. The two data sets to be joined are loaded using their corresponding schemas.

The whole project is designed as follows. First the Web Service client chooses two remote data stores and requests the join Web Service to get data from these two data stores. Then the user selects the common join attributes and result attributes. After that, the client requests the join Web Service to do the join operation and get the join result back in XML format. On the server-side, the new join Web Service invokes the data transfer Web Service to dump data from the user specified data stores and loads the dumped data into a local Relational database. After the client submits the join request along with the specified join attributes and result attributes, the join Web Service performs the join operation using a SQL query on the Relational database, dumps the join result into an XML formatted file; and returns it to the client.

3.3 Architecture

The ER Design Tool maps an ER schema into different data model schemas (Relational schema, Spreadsheet schema, XML DTD, and ODMG schema). In order to dump and load data from/to data stores with different data models, first an ER schema needs to be created using the ER Design Tool and then mapped into an application-specific XML DTD. Then the data transfer Web Service can transfer data from one remote data store to another remote data store by first dumping data from the source data store into XML Databus format (using the XML DTD for the application schema), then sending the dumped data to the target data store, and finally loading the data into the remote data store.

The new join Web Service provides two operations: *get*() and *join*(). The *get*() operation invokes the data transfer Web Service to move data from the two remote data stores to the relational data store on the server-side of the join Web Service. The *join*() operation loads data from the two remote data stores into the relational database, executes the join query, and then dumps the join result into XML format. Figure 1 shows this architecture.

3.4 Implementation class diagram

This project consists of two parts: the join Web Service and the Web Service client. The class diagram in Figure 2 includes the major classes for these two parts and the relationships between these classes.

[1] Although not used in the prototype described in this paper, we have implemented a new version of the ER Tools XML Database using the new ISO SQL/XML Standard for XML Schema instead of our original XML DTD format.

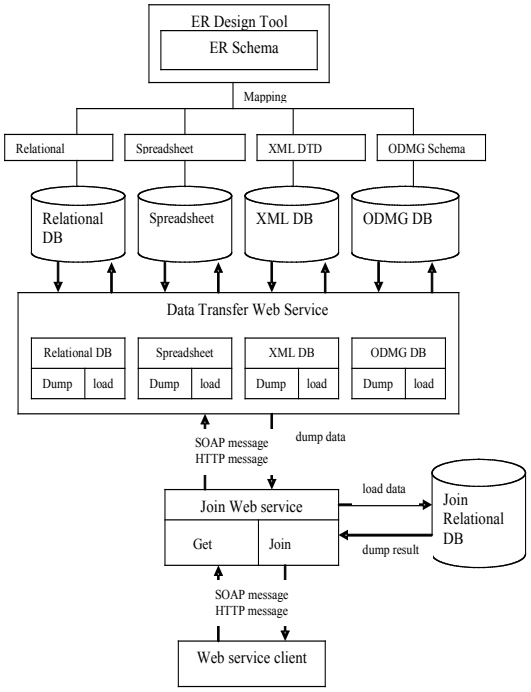


Figure 1. Web Service Join Architecture

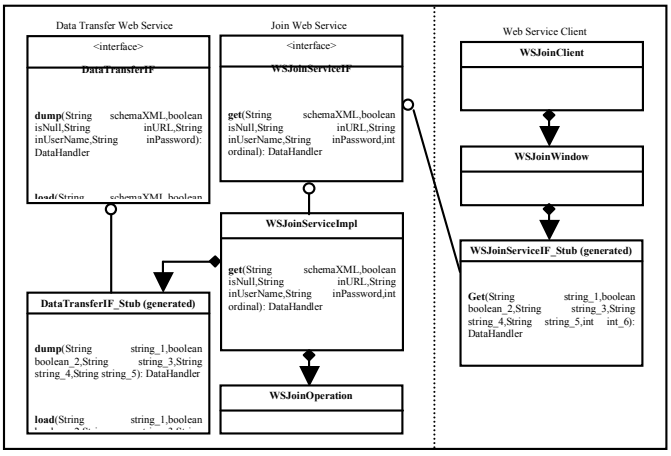
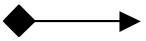
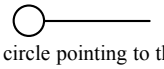


Figure 2. Class Diagram

Note:

 refers to an aggregation relationship, with the diamond pointing to the class containing an instance of the class pointed by the arrow.

 links to an interface and its implementation, with the circle pointing to the interface and the tail pointing to the implementation class.

3.4.1 Server-side classes

1) WSJoinServiceIF is an interface describing the endpoint of the join Web Service, which defines two methods: *get*() and *join*().

2) WSJoinServiceImpl is the servant class which implements the *get*() and *join*() methods of the join Web Service endpoint interface WSJoinServiceIF. The *get*() method creates an object of DataTransferIF_Stub class and invokes its *dump*() method, and returns a DataHandler referring to the data dumped from the user-specified remote data store. The *join*() method creates an object of WSJoinOperation class and invokes its *join()* method; and returns a DataHandler referring to the join result data.

3) WSJoinOperation is a class which implements the *join*() operation by loading the two join data sets obtained using the *get*() method of WSJoinServiceImpl into the default relational database; executing the join query; and dumping the join result into an XML data file.

4) DataTransferIF is the endpoint interface of the data transfer Web Service, which defines five methods: *createDB*(), *dump*(), *load*(), *generateDumpJavaCode*(), and *generateLoadJavaCode*(). Because the join Web Service is only concerned with the *dump*() method of the data transfer Web Service, the class diagram only shows *dump*() and *load*() methods in the DataTransferIF interface. The join Web Service uses the data transfer Web Service to transfer data from different data stores to the default join relational database.

5) DataTransferIF_Stub is the stub class that implements the data transfer Web Service interface DataTransferIF. Supposing that the data transfer Web server and the join Web server are not on the same machine, the join Web Service cannot directly instantiate the DataTransferServant class, which implements the data transfer Web Service interface DataTransferIF. JAX-RPC provides a tool (wscompile) to generate a stub object DataTransferIF_Stub which has the same methods as DataTransferServant. The join Web Service can use this stub object to invoke the *dump*() method of the data transfer Web Service.

3.4.2 Client Side Classes

1) WSJoinClient is the main class running the join Web Service client. It opens a main client window that allows the user to create multiple new WSJoinWindows to perform join operations.

2) WSJoinWindow is the class that allows the user to choose two remote data sources, to specify the join attributes and result attributes, to create an instance of WSJoinServiceIF_Stub, and to invoke the *get*() and *join*() methods of the join Web Service to obtain the join result.

3) WSJoinServiceIF_Stub is the stub class that implements the join Web Service interface WSJoinServiceIF. Supposing that the join Web server and the join Web Service client are not on the same machine, the client cannot directly instantiate the WSJoinServiceImpl servant class which implements the join Web Service interface WSJoinServiceIF. The client needs a stub class WSJoinServiceIF_Stub (generated by wscompile of JAX-RPC) to invoke the *get*() and *join*() methods of the join Web Service.

3.5 Implementation Plan

This project uses NCBI data sets (TSeq and GBSeq) for experimentation. Because the NIH-NCBI Web site provides its own specific XML data format and DTD, TSeq and GBSeq data sets need to be translated into XML Databus format using XSLT transform programs. Then the TSeq and GBSeq data sets can be loaded into remote data stores with different data models using the

data transfer Web Service. So far, the data transfer Web Service only wraps remote data stores with the relational data model and spreadsheet data model. Although the new join Web Service is designed in a general way, the current implementation only deals with relational databases and spreadsheets. The join Web Service invokes the *dump*() method of the data transfer Web Service to dump the GBSeq data set from the relational data store and the TSeq data set from the spreadsheet; then loads the GBSeq and TSeq data sets into the default join relational data store; executes a join query in the join relational data store; and dumps the join result into an XML formatted file. The join Web Service client specifies the remote data stores and invokes the *get*() method of the join Web Service to get the TSeq and GBSeq data sets, identifies common join attributes between TSeq and GBSeq data sets, specifies the result attributes, and invokes the *join*() method of the join Web Service to get the combination of the TSeq and GBSeq data sets. The implementation workflow diagram is shown in Figure 3.

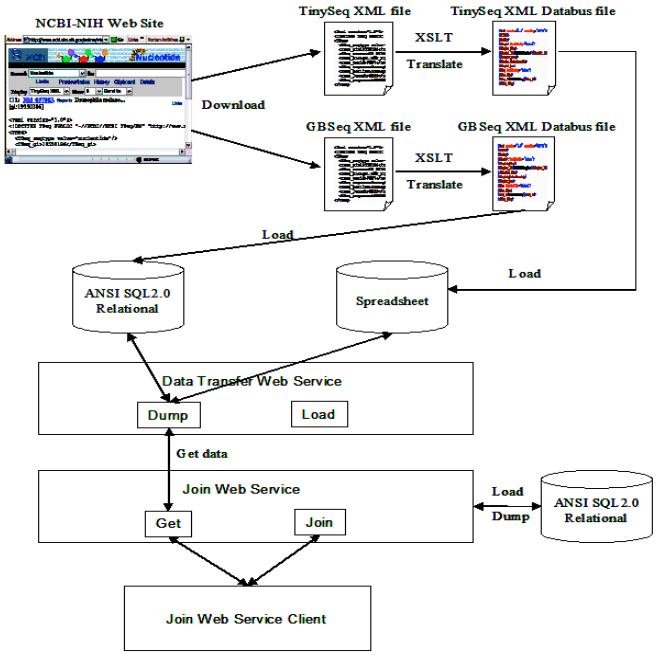


Figure 3. Implementation Workflow Diagram

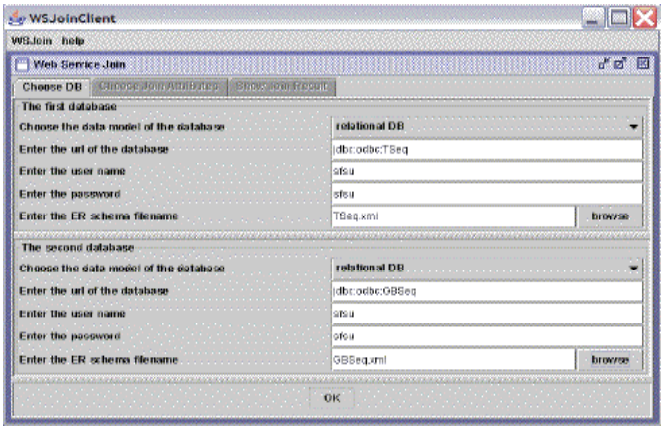


Figure 4. Interface to specify remote databases

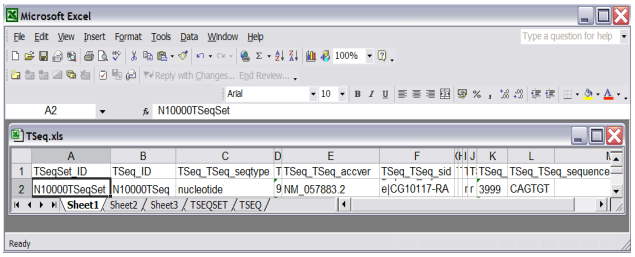


Figure 5. TSeq.xsl

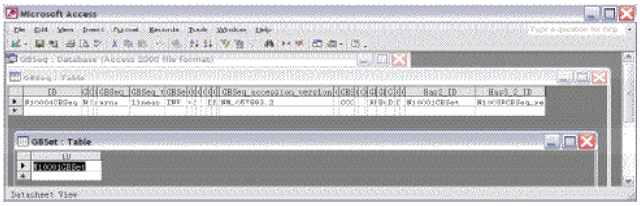


Figure 6. GBSeq.mdb

4. DATA INTEGRATION INTERFACE

Figures 4 shows the user interface for specifying the two data sets to be integrated, and Figures 5 and 6 show the two schemas to be integrated (TSeq and GBSeq). Figure 7 illustrates how the two schemas are manually integrated by specifying the common join attributes. It is also possible to select a subset of attributes to be included in the merged data set. Finally, Figure 8 shows the resulting merged data set for this small illustrative example.

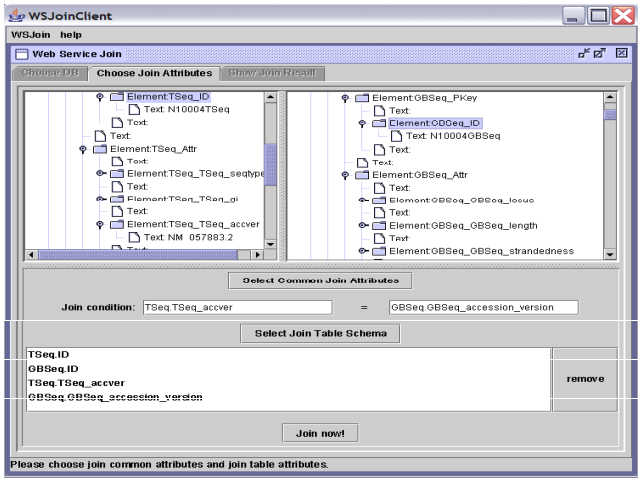


Figure 7. Specifying common attributes

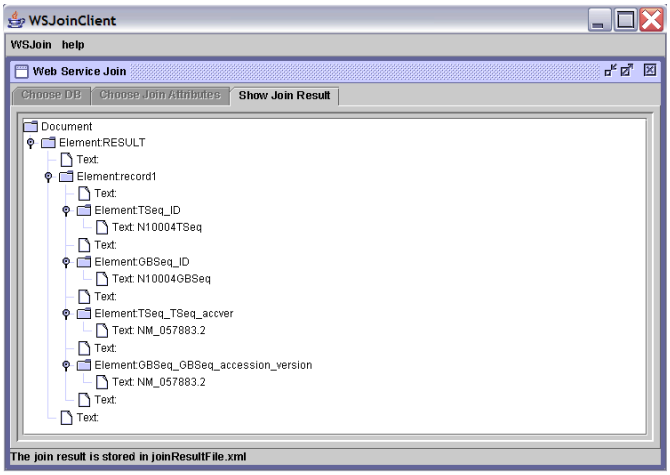


Figure 8. Result in XML

5. PERFORMANCE

The performance of software developed in this project was evaluated by measuring the execution time taken to receive the response from the join Web Service after the service client submits the requests for *get*() and *join*() operations of the join Web Service. The environment in which the join Web Service and the service client are running is a Notebook computer with the Windows XP operating system, 512MB RAM, and an AMD Athlon™ 64 processor. The data sources used for the experiments are Microsoft Excel and Access. TSeq.xls contains one table TSeq with 10 records. GBSeq.mdb contains 18 tables as follows: GBSet (with 1 record), GBSeq (with 3 records), GBAuthor (with 369 records), GBQualifier (with 62 records), and GBReference_authors (with 20 records). Figure 9 shows the execution times taken by the client to invoke the *get*() and *join*() methods of the join Web Service and to obtain the response from this service.

Because the *get*() operation of the join Web Service needs to interact with the *dump*() operation of the data transfer Web Service, the execution time takes several seconds. Although TSeq.xsl contains many fewer tables and records than GBSeq.mdb, the execution time taken to get data from TSeq.xsl is more than that taken to get data from GBSeq.mdb. This is because the Java code to perform *dump*() operations of the data transfer service is generated and executed dynamically. For the *join*() operation, the execution time is effectively based on filtering over approximately 10^6 (denormalized) records.

Table 1. Performance Measurements

Experiment	Join Web Service Method	Execution Time (milleseconds)
First time	get() (from TSeq.xls)	6391
	get() (from GBSeq.mdb)	5953
	join()	1125
Second time	get() (from TSeq.xls)	5140
	get() (from GBSeq.mdb)	4579
	join()	1312
Third time	get() (from TSeq.xls)	4735
	get() (from GBSeq.mdb)	6703
	join()	1797
Average	get() (from TSeq.xls)	5422
	get() (from GBSeq.mdb)	5745
	join()	1411

6. CONCLUSIONS

In this paper, we introduced the SFSU ER Design tools and its capabilities for database design and heterogeneous data integration. For database design, the tool provides automatic mapping of ER Schema to various implementation schemas such as Relational (ANSI SQL2), Object Oriented (ODMG 3.0), Spreadsheet (Universal Relation with associated functional dependencies) and W3C XML DTD. For heterogeneous data integration, this tool provides a data transfer web service interface to transfer data between various data stores based on a common ER Schema. This paper concentrates on our work in bio-informatics data integration using NCBI datasets. Two new web service interfaces are defined for data integration. Various heterogeneous remote data stores to be integrated are identified; users are allowed to select the join attributes from these different data stores and the attributes to be included in the result. Data integration is done by invoking a web service to dump and load both of the heterogeneous data stores to a common Relational site and then performing SQL joins based on user specified attributes. The result is available in XML format. Our prototype implementation is operational and exhibits relatively good performance.

7. ACKNOWLEDGMENTS

Conference registration support from the SFSU Center for Computing for Life Sciences (http://cs.sfsu.edu/ccls/index.html) is gratefully acknowledged. Our thanks also to all of the students

who have contributed to the SFSU ER Design Tools, both as system developers and as end users.

8. REFERENCES

[1] S. Vincent, Heterogeneous Data Transfer Using Web Services. M.S. Project Report. SFSU Computer Science Department Technical Report: SFSU_CS_TR_05_05 (2005).

[2] S. Prasad, a Web Service Implementation of the ER Design Tool. M.S. Project Report. SFSU Computer Science Department Technical Report: SFSU_CS_TR_05_06 (2005).

[3] R. Ramakrishnan, et al, Database Management Systems. McGraw-Hill Science, 3rd edition (2002).

[4] P. Hong, S. Zhong, et al, Towards Ubiquitous Bio-Information Computing: Data Protocols, Middleware, and Web Services for Heterogeneous Biological Information Integration and Retrieval. Proceedings of the Fourth IEEE Symposium on Bioinformatics and Bioengineering (BIBE'04) 0-7695-2173-8/04 (2004).

[5] S.B. Davidson, et al, K2/Kleisli and GUS: Experiments in integrated access to genomic data sources. IBM Systems Journal, 40(2): p. 512-531 (2002), http://www.research.ibm.com/journal/sj/402/davidson.pdf

[6] T.Y. Jen, using XSLT to Translate NIH XML Data Files into ER Tools XML Data Files. M.S. Project Report. SFSU Computer Science Department Technical Report: SFSU_CS_TR_04_40 (2004).

[7] E. Armstrong, Working with XML_ The Java API for XML Processing (JAXP) Tutorial. Date retrieved: 11/09/2006. http://java.sun.com/WebServices/jaxp/dist/1.1/docs/tutorial/index.html

[8] K. Topley, JAVA Web Services in a Nutshell. O'Reilly (2003). http://java.sun.com/WebServices/jaxp/dist/1.1/docs/tutorial/index.html

[9] E. Armstrong, J. Ball, et al, the J2EE™ 1.4 Tutorial (2005). http://java.sun.com/j2ee/1.4/docs/tutorial/doc/

[10] D. Webber, D. and A. Dutton, Understanding ebXML, UDDI, XML/EDI (2000) http://www.xml.org/xml/feature_articles/2000_1107_miller.shtml

[11] Web Services. Apple Computer, Inc (2002).

[12] Yan Liu, Web Services Joins Using ER Design Tool XML Databus, M.S. Project Report. SFSU Computer Science Department Technical Report: SFSU_CS_TR_06_09 (2006).

[13] SFSU ER Design Tools Web Site: https://dbsgrad.sfsu.edu/~erdesigntools