

Integration of Data from Heterogeneous Biological Databases

Jon Deron Eriksson & Mei-Ling L. Liu

Computer Science Department

California Polytechnic State University, San Luis Obispo,

California, U.S.A.

mliu@csc.calpoly.edu

Abstract

The Internet has become a unified transport infrastructure for cooperative computing and information sharing. This paper describes the prototype for a framework contributing to cooperative information sharing in the emerging field of bioinformatics. The work makes use of a number of newly emerged technologies in distributed computing to provide a powerful yet flexible method of integrating data from heterogeneous biological databases.

1. Background

Today, hundreds of public biological databases are accessible via the Internet [NAR2001, DBC2001]. However, taking advantage of data stored in heterogeneous biological databases can be a difficult, time-consuming task for a multitude of reasons. These reasons include the vast volume of biological data, the growing number of biological databases, the rapid rate in the growth of data, the overabundance of data types and formats, the wide variety of bioinformatics data access techniques, database heterogeneity, and the interdisciplinary nature of bioinformatics.

In the course of his/her work, a scientist may query a biological database, visually scan the results of this query, locate the data of interest, and subsequently use this data in a query submitted to another biological database. Performing manual multi-database queries in this manner can be a time-consuming process, especially when large amounts of data are involved, as is often the case in biological research. Therefore, a need exists for automated data integration from multiple, heterogeneous databases. Automating biological database data integration can speed up the discovery of new medications and the introduction of these drugs to market, with potentially wide ranging benefits to mankind.

This work presents a proposed solution to this problem, involving the use of a number of enabling technologies. The solution entails the creation of a database federation that allows queries to be performed using a user-friendly client application. The client application accesses the database federation system via a Java servlet [SUN2002]. The data integration system also makes use of CORBA (Common Object Request Broker Architecture) technology [OMG2002]. The interoperability of CORBA objects allow the data integration system to be located on a single workstation or distributed on multiple workstations. In addition, the language independence of CORBA allows the data objects to be implemented in different programming languages. Adding to the interoperability, XML is used as a meta-language for database queries. Although there are existing works in bioinformatics which employ CORBA technology, such as the CORBA-based access to the Radiation Hybrid database (RHdb) [LIJ1998, SPI2000], there remains a need to develop versatile and user-friendly solutions to biological data integration.

2. The System Architecture

The proposed federated data integration system consists of three types of collaborative, distributed components: a *system access* object, a *query processing* object, and multiple *database access* objects. These objects intercommunicate via CORBA, and employ XML as a meta-language for data representation.

- The *system access* object provides the interface for a client to communicate with the data integration system, and is responsible for returning query results to the client.
- The *query processing* object is responsible for determining which databases need to be contacted in order to perform a query, and it is also responsible for processing the results of such queries. Each *database access*

object interacts with a specific database, typically via a web server.

- The *database access* object accepts a query from the query processor and formats the query in a form specific to its database. The *database access* object also receives results from database queries and formats these results into a form understandable to the data integration system.

An overview of the architecture of the prototype system is presented in Figure 1. The system comprises four objects: a *system access* object, a *query processing* object, and two *database access* objects. The *database access* objects support querying to two popular biological databases: the EMBL nucleotide sequence database [EMB2002] and the PubMed biological abstract database [ENT2002]. Thus, the system allows integrated access to two heterogeneous databases.

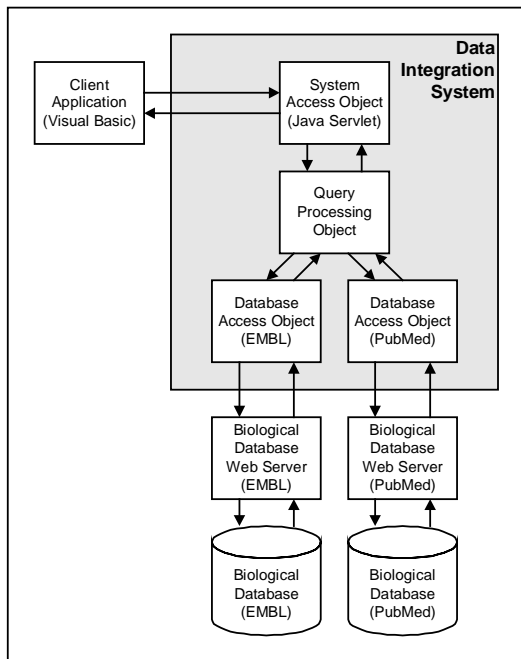


Figure 1: Federated Data Integration System Overview

The *system access* object provides the “front end” of the system, and is implemented as a Java servlet that accepts queries from clients via the Hypertext Transfer Protocol (HTTP) GET and POST methods [HYP1999]. The *system access* object provides a layer of abstraction that separates the client application from the details of the “back end” of the system. The three-tier

hierarchy allows the system internals of the data integration system to be modified without affecting the client applications. For example, the data integration system can be implemented using an alternate distributed computing technology, such as Java RMI (Remote Method Invocation) [HAR2000], without affecting how the clients access the system.

The *system access* object accepts the client’s query and formats the query into a well-formed XML string, which is sent to the *query processing* object. The *query processing* object decides which database or databases need to be queried in order to complete the query based on the query type, and it in turn contacts the appropriate *database access* object or objects using query strings embedded in an XML message.

Database access objects perform queries of web-accessible biological databases using the HTTP GET method. Each type of *database access* object is customized for data access to a specific biological database, using the data format and access method specific to that database. A *database access* object receives an XML-formatted query from the *query processing* object and maps the query into the specific format required by the database that it accesses. The *database access* object issues the query to the database, and the result is translated into an XML message that is sent back to the *query processing* object. Based on the query type, the *query processing* object may make use of the result to issue a subsequent query, or it may send the response back to the *system access* object, which returns the results to the client.

An example of a multi-database query that can be performed by the system is shown in Figure 2. In this example, the client requests the system to perform a query that returns all PubMed abstracts that are associated with an EMBL DNA sequence entry.

The client submits the query type and an EMBL accession number to the system via the *system access* object using an HTTP request (step 1). The *system access* object formats this query in XML and passes this query string to the *query processing* object by invoking a CORBA method of the object (step 2). The *query processing* object examines the query type and determines that the first database to be queried should be the

EMBL sequence database. Therefore, it creates an XML-formatted query requesting an XML-formatted EMBL entry based on an EMBL accession number, and issues this query to the EMBL *database access* object via a call to a CORBA method (step 3).

In the invoked method, the EMBL *database access* object extracts the accession number from the parameter, and performs a query of the EMBL sequence database using the accession number by issuing an HTTP GET method to the EMBL web server (step 4). The EMBL web server returns an EMBL sequence string to the EMBL *database access* object (step 5), which parses the string and maps it into a hierarchical XML representation. This XML representation of the EMBL entry is then returned to the *query processing* object (step 6).

Upon receiving the XML representation of the EMBL entry, the *query processing* object extracts all PubMed citation unique identifiers referenced in the entry, creates an XML-based query string with those identification numbers, and passes this string to the PubMed *database access* object via a CORBA method call (step 7). In accepting the method call, the PubMed *database access* object uses the identification numbers extracted from the query string to generate a query in the form of an HTTP GET method to the PubMed web server (step 8). In return, the *database access* object receives the abstracts referenced by the unique identifiers in plain text format (step 9). These abstracts are then mapped to an XML representation, which is returned to the *query processing* object (step 10). The query processor receives the query result and returns these abstracts to the *system access* object in the return string of the query processor method that the *system access* object originally invoked (step 11). Finally, the *system access* object returns the abstracts to the client application (step 12), thus completing the multi-database query.

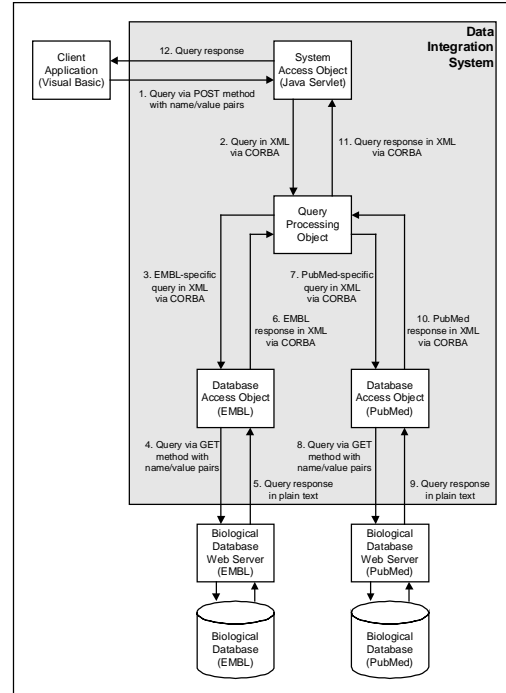


Figure 2. Multidatabase Query Example

Figure 3 is Unified Modeling Language (UML) class diagram which describes the hierarchy of object classes employed in the data integration system.

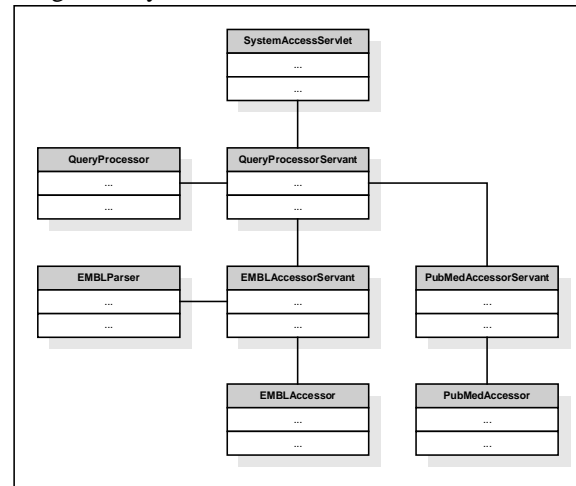


Figure 3: UML Class Relationship Overview of the Data Integration System

The *QueryProcessor*, *EMBLAccessor*, and *PubMedAccessor* are CORBA object servers. Each server instantiates a servant and registers this servant with an ORB. Each servant is responsible for implementing the methods of a component's CORBA interface. After

registering a servant's name with a naming service, a server waits for clients to invoke servant methods.

Respectively, the *QueryProcessor* instantiates a *QueryProcessorServant*, the *EMBLAccessor* instantiates an *EMBLAccessorServant*, and the *PubMedAccessor* instantiates a *PubMedAccessorServant*. The *QueryProcessorServant* invokes methods provided by the *EMBLAccessorServant* and the *PubMedAccessorServant*, and the *SystemAccessServlet* invokes methods provided by the *QueryProcessorServant*. The *QueryProcessorServant* and *SystemAccessServlet* are CORBA clients, since they invoke methods on CORBA objects. The *QueryProcessorServant* is itself also a CORBA object.

The *EMBLAccessorServant* instantiates an *EMBLParser* for queries requiring XML parsing of EMBL strings. This parser takes an EMBL string entry and converts it into an XML format. At the time of this writing, the EMBL web server can return accession number query results in two XML formats – AGAVE and BSML. However, the XML format provided by the *EMBLParser* class used in our system provides a more comprehensive representation of the data and structure of EMBL strings.

An example EMBL string entry and the *EMBLParser*'s XML representation of this entry are shown in the Appendix.

3. Prototype Implementation

A prototype system of the proposed architecture was implemented. This prototype consisted of a *system access* object, a *query processing* object, and two *database access* objects (the EMBL database accessor and the PubMed database accessor). The data integration system and accompanying client application were developed on personal computers running the Windows NT and the Windows 2000 operating systems. All system objects were coded in the Java programming language using JDK 1.3. The client application was implemented in Visual Basic to run on the Microsoft Windows operating systems. Other clients to this data integration system can easily

be constructed in other languages to run on other platforms. The servlet engine employed for the prototype was JSWDK 1.0.1 (JavaServer Web Development Kit 1.0.1). Java IDL was used for CORBA implementation. A screenshot of the client application during a test run is shown in Figure 4.

Conclusion

This paper presents a prototype system which makes use of enabling technologies such as CORBA, Java, and XML to provide a powerful yet flexible method of integrating data from different biological databases. The system was designed to allow it to be extendable to support integrated access to any number of heterogeneous databases, and should be a useful tool for scientists and researchers in the field of or in need of bioinformatics.

References

- [DBC2001] "DBCAT," <http://www.infobiogen.fr/services/dbcat>
- [EMB2002] "The EMBL Nucleotide Sequence Database," <http://www.ebi.ac.uk/embl/>
- [ENT2002] "Entrez-PubMed," <http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?db=PubMed>
- [EXT2002] "Extensible Markup Language (XML) 1.0 (Second Edition)," <http://www.w3.org/TR/REC-xml>
- [HAR2000] Elliotte Rusty Harold, "Java Network Programming," O'Reilly & Associates, 2000.
- [HYP1999] Hypertext Transfer Protocol - HTTP/1.1 - Draft Standard RFC 2616, <http://www.ietf.org/rfc/rfc2616.txt>
- [LJ1998] Philip Lijnzaad, Carsten Helgesen and Patricia Rodriguez-Tomé, "The Radiation Hybrid Database", *Nucleic Acids Research* 26(1), pp. 102-105, 1998.
- [MCL2000] Brett McLaughlin, *Java and XML*, O'Reilly and Associates, Inc., 2000.
- [NAR2001] "Nucleic Acids Research 2001 Molecular Biology Database Collection," <http://nar.oupjournals.org/cgi/content/full/29/1/1/DC1>
- [OMG2002] "The Object Management Group's CORBA Website," <http://www.corba.org>
- [RAJ1999] Pethuru Raj and Naohiro Ishii, "Interoperability of Biological Databases by CORBA," *Proceedings of the 1999 International Conference on Information Intelligence and Systems*.
- [SPI2000] Anastassia Spiridou, "A View System for CORBA-Wrapped Data Sources," *Proceedings of the IEEE Advances in Digital Libraries 2000 (ADL 2000)*.
- [SUN2002] "JAVA™ SERVLET TECHNOLOGY", <http://java.sun.com/products/servlet/>

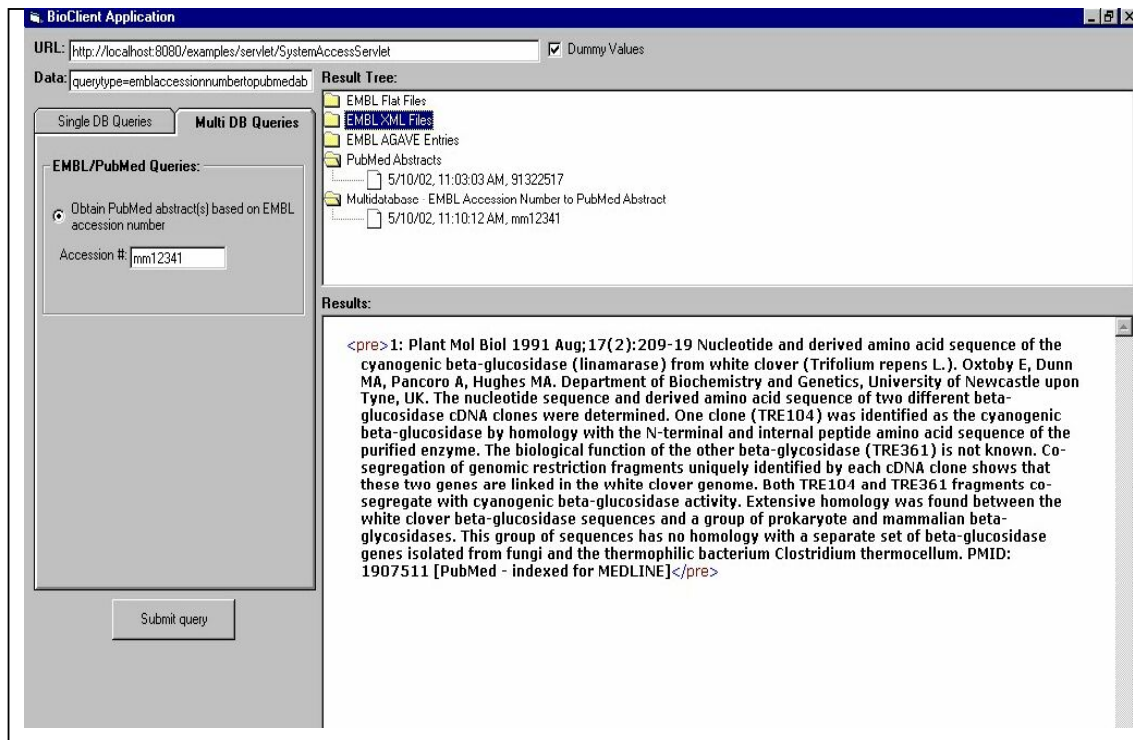


Figure 4: Client Application Screenshot

Appendix

Due to space limitation, the sample EMBL string entry and its corresponding XML representation cannot be provided here, but will be available upon request.