J3DV: A Java-Based 3D Database Visualization Tool

Xiang Fang
110 Rogers Rd. Apt.N104
Athens, GA 30605
Tel: (706) 369-8579
xfang@cs.uga.edu

John A. Miller
415 GSRC, Computer Science Department
The University of Georgia
Athens GA, 30602-7404
Tel: (706) 542-3440
Fax: (706) 542-2966
jam@cs.uga.edu

Jonathan Arnold
Genetics Department
The University of Georgia
Athens, GA 30602
Tel: (706) 542-1449
arnold@arches.uga.edu

John A. Miller is the contact person.

SUMMARY

Data visualization helps users process, interpret, and act upon data in large data storage. In this paper, we present a Java-based 3D database visualization tool, J3DV. It successfully solved the problem of data management faced by many other visualization systems by integrating multiple data sources with the visualization tool. It utilizes two-level mapping to transform the data into intermediate data, which can be used to render graphs, and which offers better performance with a two-tier cache.

This visualization tool presents a sound framework, which has good extensibility for plugging in new data sources, supporting new data models and visual presentation types and allowing new graph layout algorithms.

INDEX WORDS: Database visualization, Data model, Mapping, Java 3D, Bioinformatics.

1. Introduction

Modern hardware and database technology has made it possible to store gigabytes of information in databases. However, it can be difficult to utilize fully the information in a large database with complicated structures. Many methods, such as data mining, have been proposed and studied to help users better understand and analyze the information. Database visualization is one of the effective solutions to this problem. Database visualization becomes more appealing when handling large data sets with complex relationships because information presented in the form of images is more direct and easily understood by humans. Database visualization has been widely used in many scientific research areas. Current visualization systems, such as MineSet [18], provide users powerful tools to display and manipulate data. However, they offer insufficient data management capability. In this paper, we introduce a database visualization tool ---

J3DV, which improves this weakness by providing integration of data sources with visualization and enhances performance with a two-tier cache.

1.1 Database Visualization

"Database visualization transcends traditional computing disciplines by creating visual imagery with scientific data" [16]. It is a procedure performed by computer programs, which integrate data from different sources, and then transform data into images, which result in knowledgeable action. According to Lang *et al.* [15], the database visualization process consists of the following steps:

- Retrieving data from data sources. The data sources can be of different types (*e.g.*, relational databases, object-oriented databases or file systems). In addition, the location of data sources can be on a local machine or distributed over the network.
- Mapping. This step transforms the original data into a geometric representation of the data, providing a visual representation.
- Rendering and displaying. This step produces and displays images.

Database visualization is data centric. It makes data the core of the software system development process. Understanding the data in data sources is the basic requirement of database visualization. According to Berson *et al.* [6], the data resources for database visualization fall into three categories, namely, computer simulations, statistics, and data gathered from natural systems. These data undergo many transformations from an original state to a displayable state. These data transformations must be presented at the user interface, and include data storage, retrieval, and filtering [16].

Database visualization should not only show pictures but also allow the user to explore the data in the large data sources. Goldstein *et al.* [7] proposed that a visualization tool usually has three functions:

1) Data visualization, which may include transforming data into a displayable state and rendering the image based on those data.
2) Data manipulation, which may include retrieving data, removing unwanted attributes of data and reorganizing data.
3) Data analysis, which may include statistical reports on data and summarization of data.

The data source and visualization system have different data models. A database visualization tool must make a connection between the data source data model and the visualization data model. Some methods has been proposed and studied. For example, Lee [17] described a database management–database visualization integration, which uses the view concept of relational databases to link database and visualization systems. However, this method is not general enough to be applied to all data sources. To solve this problem, a comprehensive data model is needed to make the connection. “A data model is an abstraction of the data” [1]. Different visualization applications use different data models. It is not feasible to create a single data model for all visualization systems.

Database visualization uses different visual representations to express multivariate data. How to present the data with appropriate visual representations is important. A number of database visualization techniques have been studied and developed, such as 3D imaging, colorization and animation [22]. A variety of presentation methods have been employed. For example, the MineSet product of Silicon Graphics Inc. has a series of

visualizers, such as Map Visualizer, Tree Visualizer, *etc.*. These visualizers can be used to view data with different structures.

When discussing database visualization, it is necessary to mention a closely associated field, data mining. Data mining is the process of extracting interesting knowledge from large amounts of data stored either in databases, data warehouses, or other information repositories [9]. According to them, data visualization and data mining can be integrated in several ways, such as data mining result visualization, data mining process visualization or interactive visual data mining.

1.2 Current State

Compared with other areas of computer science, database visualization has a relatively short history. The recent emphasis on this research area emerged about fifteen years ago [22]. Examples of some early visualization systems can be found in the work of Asimov [2], Baecker [4], and Beddow [5]. The first IEEE workshop dedicated to database visualization was held in 1993. Some advances have been achieved in this field since then. Many visualization tools have been developed, for commercial and research purposes. For example, the MineSet of Silicon Graphics Inc. and the AVS/Express [3] of Advanced Visual System Inc. are mature commercial visualization products. Visage [25] is a research project carried out at the Carnegie Mellon University. In addition, the Open Visualization Data Explorer [20] of IBM is an open source visualization tool.

However, it has been pointed out that current data visualization environments offer insufficient support for data source management and data interrogation methods. Compared with the visual representation of data, system functions are underdeveloped.

While the issues of graphical modeling and rendering are studied sufficiently, the methods of data modeling and data retrieving are not efficient [27].

With the increasing complexity of data, the data management problem becomes more apparent. According to Arya *et al.* [1], there are two options to deal with this problem. The first option is to incorporate data source management capability into visualization systems. Some visualization systems adopt this option to handle the data management problem. However, most of these systems are not general-purpose solutions, but only applicable for some specific visualization systems. Other visualization systems adopt the other option to integrate visualization tools with database systems. However, the challenge of this approach is how to define an efficient and sound interface between the visualization system and the data sources. Many solutions have been proposed and studied [1].

1.3 Motivation and Goals

In the present paper, a Java-based 3D database visualization tool (J3DV) is discussed. This tool is an integral component of the Protein-Protein Interaction Workflow, which in turn is an important part of the Fungal Genome project. The Fungal Genome project is a multi-institutional project with the goal to create high-resolution physical maps and determine the complete genome sequences of important fungal organisms. The Protein-Protein Interaction Workflow is currently being developed at the University of Georgia.

Protein-protein interactions play a key role in the structural and functional organization of a cell. The interactions of novel proteins with known proteins provide important clues to the understanding of the functions of these novel proteins. Many

protein-protein interaction experiments have been carried out. For each experiment, 96 preys (proteins) and 96 baits (proteins) are grouped together to see if there exist interactions among a given pair of proteins [11]. These data are usually stored in a database or file system.

One task of the protein-protein interaction mapping workflow is to find the protein-protein interaction network [11,26]. The motivation of J3DV is to extract information from a database and generate the protein-protein interaction network.

Although the motivation for this system is to view the protein-protein interaction network, we do not want to build a special purpose tool with limited functions. During the design and implementation of the system, we set up the following goals:

- Generic. The tool should be application domain independent. In other words, it can visualize data from other domains.
- Integrated. This system should be better integrated with data sources, so that it can offer efficient data access and management.
- Extensible. This system should have a sound framework, so that we can easily add more visual representation support.
- Pluggable. Scientific data can exist in any form, some in databases, some in specially formatted data files. It should be easy to plug in new data sources.
- Easy to use. Most end-users have little programming experience, so ease of use is very important.
- Portable. The tool should be deployable across platforms.

The outline of the rest of this paper is as follows. Section 2 introduces the technological infrastructure of J3DV and its architecture. Section 3 discusses the system integration issues and compares J3DV with other visualization tools. Section 4 deals with data modeling and mapping. Section 5 walks through the use of J3DV. Finally, Section 6 concludes the paper and discusses future work.

2. Architecture and Technological Infrastructure

In this section, we introduce J3DV in depth. Before that, we briefly review the technology utilized in the visualization tool.

2.1 Technological Infrastructure

Modern software technology provides much more computing power so that difficult problems can be solved more easily today. Java is such a software platform. Its central promise, “write once, run anywhere”, has earned a reputation in today’s information technology. The J3DV system is implemented in Java. The primary factors for choosing Java are that it is platform-neutral, robust, and supports applications deployed over heterogeneous network environments. In addition, the popularity of the Java language itself is also a factor. We briefly review the technology used in this system.

Database visualization is data-centric, so integration of data sources with a visualization tool is very important. Also, a visualization tool should be able to import data from different data sources, instead of some specific data source. From the beginning of this project, we have focused on finding an efficient and general-purpose approach to access multiple data sources. The Java Data Objects (JDO) specification[13] provides such an approach. It was proposed by Sun Microsystems Inc. to handle storage issues for Java objects in heterogeneous storage. It provides transparent access to different data sources, and is extensible to plug in new data sources. At the time this paper is being written, the JDO specification is still a document and no official implementation has been released. We implemented parts of the specification, which were relevant to our project (see [3] for the details of the JDO specification).

The Java language features are also useful in J3DV's mapping procedure. The mapping procedure transforms data from data source into intermediate data, which can be used by the visualization system. Collecting data type information during run-time is the key to the mapping and generating of intermediate data. Reflection capabilities provided by Java can be used to solve this problem. It can help collect data object's run-time type information (RTTI), which is important in transforming original data into intermediate data.

Java 3D technology is the key to our client-side implementation. It is the result of a joint collaboration among Silicon Graphics, Inc., Intel, Apple Computer and Sun Microsystems. It draws its ideas from these companies' existing products, such as OpenGL [21] and incorporates new technology. In short, Java 3D can optimize the underlying hardware for better performance. It utilizes geometry compression to reduce the potential bottleneck in network bandwidth.

Java Remote Method Invocation (RMI) technology establishes a connection between the client and the server. An RMI distributed application can obtain a reference to a remote object via an RMI registry, which keeps the information about the registered remote objects. The server registers a remote object in the registry, which will associates a name with that remote object. The client can reference the remote object by looking up the remote object's name in the server's registry and then invoking a method on it. The RMI system uses hypertext transfer protocol (HTTP) to load class byte-codes from server to client.

The Java Naming and Directory Interface (JNDI) is another technology, which connects the client and sever. JNDI is an application programming interface that provides

naming and directory functionality to applications. It is independent of any specific directory service implementation, such as DNS and LDAP.

2.2 Architectural Overview

One of the key issues of database visualization is how to integrate the data sources and the visualization tool. The method of integration decides the architecture. Currently, three types of architectures have been identified to support the integration [1]:

1) Close-coupled architecture. All the components of the system, including data sources and visualization tool, reside on one machine. This architecture is best for single user, frequent transaction scenarios.
2) Client-server architecture. Two or more machines connected over a network play different roles. Data sources act as servers by providing services, such as data access, data transformation, and computing. Others act as clients, which the end users interact with.
3) Distributed asynchronous process communication architecture. The visualization system does not access data sources; there are other modules standing between the visualization and data sources.

The latter two architectures both follow the client-server model and allow the distribution of services, interoperability, and multiple clients using the services. The third architecture can reduce data access time. Most of the popular visualization tools follow the client-server model. For example, Open Visualization Data Explorer employs an extended data-flow-driven client-server execution model. The client process uses a graphical user interface. The server process does most of the computation and typically

resides on a different machine. Medium or fine-grain configurations of symmetric multi-processor servers provide significant scaling for the server process when applied to larger, more complex data sets.

J3DV adopts the distributed asynchronous process communication architecture. This is because the client does not access data source directly. Instead, the server stands between the data sources and the client, and the server pipelines and caches the data. Figure 1 shows its architecture.

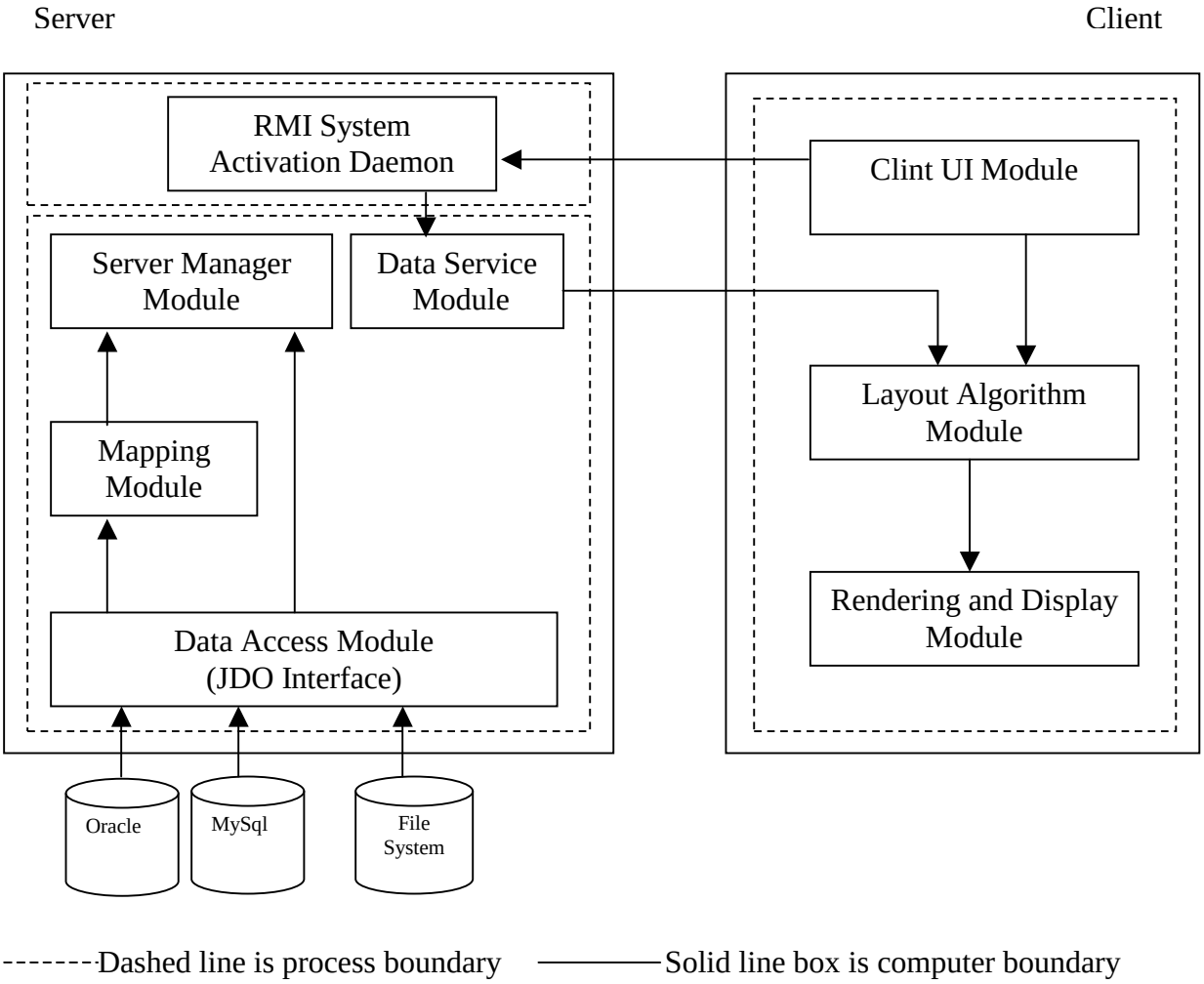

Figure 1. The Architecture of J3DV System

2.3 Server-Side Modules

The server performs several tasks, including integrating multiple data sources with visualization, transforming the original data into an intermediate form that can be utilized by visualization clients, and caching and pipelining the mapped data. The server consists of four modules: Server Manager Module, Data Access Module, Mapping Module and Data Service Module.

2.3.1 Server Manager Module

Server Manager Module provides a friendly graphical user interface, through which the users can interact with the system to decide the data source, view schema of data sources, choose a data set and specify mapping relationship. The users using the server must have a better knowledge on the database.

J3DV provides visualized metadata of the data sources. This provides at least two advantages. First, it allows the user to visualize the internal structure of a data source, and navigate across the database to locate interesting data sets. Second, the metadata includes table and attribute information, which can be referenced during mapping. Section 5 includes some screen shots of the Server Manager module.

2.3.2 Data Access Module

The data access module is responsible for retrieving and filtering raw data from the data sources and transforming raw data into Java objects. To facilitate transparent access to multiple data sources and plugging-in new data sources, we choose the Java Data Object (JDO) specification as the backbone of this module. JDO specification

enables pluggable implementations of data sources into applications by defining a set of interfaces between data sources and application [13]. These interfaces cover connection, storage, query, transaction and other data access aspects. In our project, we implemented the parts of JDO that are relevant to our project. The implementation includes connection, storage and query interfaces.

Currently, J3DV supports not only databases, such as Oracle and MySql, but also local file systems as its data sources. The databases can be distributed over a network. When databases are used as data sources, we use JDBC to access data sources. This makes it easy to use other database management systems (DBMS) that support JDBC. When local file systems are used as data sources, we implement all the data access methods, such as reading metadata and retrieving data sets.

2.3.3 Mapping Module

The mapping module gets Java objects (input Java objects) from the data access module, and then transforms them into intermediate Java objects based on the intermediate data model. These intermediate Java objects can be used to render 3D graphs by the client. The data model will be discussed in more depth later (see section 4.2).

In J3DV, in order to create a mapping, a user needs to provide two things. The first is the mapping correspondence. It specifies which attribute of an input Java object is mapped to a given intermediate Java object attribute. The second is mapping rules, which defines how an input Java object attribute, which is specified in mapping correspondence, is mapped into the corresponding intermediate data attribute. To facilitate the mapping

process, J3DV provides a mapping rule generator, which helps a user specify mapping rules. Figure 7 is the screen shot of the mapping rule generator.

2.3.4 Data Service Module

As we will discuss in section 2.4, the server of J3DV uses a two-tier cache mechanism to improve the performance. In the second tier cache, the intermediate data that were mapped previously are cached. The data service module manages these intermediate data cached in the second tier cache and provides the client with these previously mapped data. This module provides remote methods, which can be invoked by remote processes, and executed on demand, *i.e.*, when a client sends a data request, the RMI activation daemon will trigger the corresponding methods in this module. It, in turn, sends the requested intermediate data back to the client.

2.4 Communication Between Client and Server

The communication between the server and the client is through the Java Naming and Directory Interface (JNDI) or a Java RMI activation daemon. The server uses a two-tier cache mechanism to save intermediate data for clients. The first tier cache is an in-memory cache, which caches the newly mapped data of the server. The second tier is a disk-based cache that caches previously mapped data.

The first tier cache is accessed through JNDI, while the second tier cache is achieved through the RMI activation system daemon. When the server creates a new mapping and generates the intermediate data, it saves these intermediate data in both caches and registers the data with a mapping name in both JNDI and RMI activation

system daemon. When the server process shuts down, it has to un-register the data in JNDI, *i.e.*, remove the registered mapping name from JNDI. However, the RMI activation daemon still has the registered name, so it is still able to provide data service to clients.

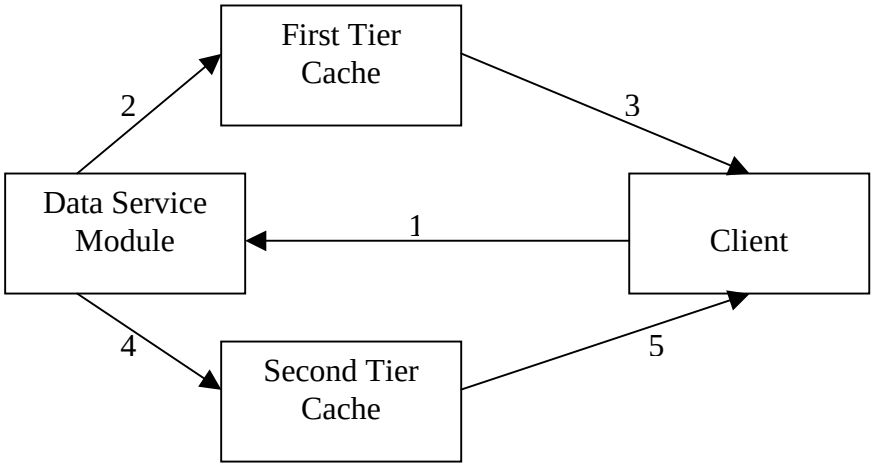

Figure 2. Cache Organization and Information Flow

Figure 2 shows the cache organization and the information flow when the client sends a data request. When a client sends a data request with mapping name to the server (represented by arrow labeled 1), the data service module will handle the request. First, it will check if the specified intermediate data is in first tier cache by looking it up through JNDI (arrow 2). If the data is found in the first tier, it will be sent to the client (arrow 3) and the data service module does not check second tier cache at all. Otherwise, the data service module will check second tier cache by looking up the mapping name on disk (arrow 4). If the mapping data is found, it will be sent to the client (arrow 5). However, if the requested data is not in either of the caches, the server has to do the requested mapping and then send the intermediate data to the client.

The two-tier cache mechanism provides much better performance. Table 1 shows the results of comparison tests. The comparison tests consist of three cases; one uses the first tier cache only, another uses the second tier cache, and the third does not use any cache. We applied the same data load, *i.e.*, the data requested by the client, on all three cases. We measure data loading time, which is elapsed time between the point the client sends out a data request and the point the client gets data from the server. We tested with two different data loads, with data load 1 having 257 Java objects (nodes and edges), and data load 2 having 415 Java objects. Although the measured time may vary with some factors, such as network topology, network traffic, and system performance of the server machine, we can still find that the caches provide much improved performance, while the first tier cache offers better performance than the second tier does.

Loading Time (Milliseconds)	First Tier Cache (Memory)	Second Tier Cache (Disk)	No Cache Used
Data Load 1	157	224	753
Data Load 2	229	537	1167

Table 1. Cache Performance Tests (times in nilliseconds)

The overall communication framework of J3DV has several advantages. First, the server does not need to run all the time to wait for a client requests; it only needs to register its data service, and this data service is only executed “on demand”, *i.e.*, upon the request of a client. It saves system resources and computational time. Second, it has very good extensibility. In the future, when we support more visual representations, we can register more data services with the daemon; for clients, they just need to tell the daemon which service they want. So this communication framework is also extensible.

2.5 Client-side Modules

The client gets the intermediate data from the server, applies a layout algorithm, transforms the intermediate data into displayable data, and displays the data graphically. The client consists of three modules: the Client User Interface Module, Layout Algorithm Module and Graph Rendering and Display Module.

2.5.1 Client User Interface Module

The Client User Interface (UI) module provides a graphical user interface for users to interact with J3DV. It helps a user to request mapping data from a server, and also provides some interaction methods for the user to control the display. These methods include position control, rotation control and other interaction methods. Figure 9, 10 and 11 show the windows provided by the client UI module.

2.5.2 Layout Algorithm Module

The J3DV displays the data in the form of graphs. A graph consists of a set of nodes and edges. “The most often used approach to graph drawing is to set the position of each vertex of the graph in such a way that the final graph would satisfy some predefined requirements such as minimizing edge crossings, symmetry, and uniform vertex distribution” [30]. Zhang [30] conducted thorough work in this research area.

The Layout module may contain several layout algorithms. In J3DV, to allow maximum flexibility on graph drawing and manipulation, we defined the graph layout algorithm interface, which consists of several graph-drawing methods. The user can plug

in their own layout algorithm by implementing the layout algorithm interface. Tian [24] gives more detail on this topic.

2.5.3 Graph Rendering and Display Module

The Graph Rendering and Display Module initializes the Java 3D environment and transforms the mapped data into displayable data, which can be directly used to render 3D graphs. The user can manipulate the graph and use the search function to locate a node or clusters of nodes. Tian [24] and Zhang [30] described the client architecture and implementation in depth.

3. System Integration

A number of issues need to be addressed in order to enable data source-visualization integration. According to [1], these issues fall into five categories: modeling, importing and exporting, querying, distribution and heterogeneity. In this section, we focus on importing and exporting, querying, distribution and heterogeneity. Modeling will be the topic of the next section.

3.1 Importing and Exporting

Flexible and efficient input and output mechanisms for data in the data source are the goal for our visualization system. A sound visualization system should provide users with the flexibility of accessing and using data as efficiently as possible. This can be made possible by providing the interoperability of various visualization systems and data sources, which leads to the need for data transformation. As for when and where this

transformation occurs, different visualization systems provide different solutions. For example, IBM's OpenDX uses General Array Import [20] to import data from data files. The General Array Importer uses a "header file" to describe the structure and location of the data to be imported. This file consists of keyword statements that identify important characteristics of the data (including grid structure, format, and data type, along with the path name of the file containing the data). For OpenDX, the data transformation occurs when a user creates the header file. However, in order to create the header file, a user must have very good knowledge of the data in the data file and the syntax of the header file, which can be a real challenge for users with little programming experience. As for data exporting, OpenDX supports various image export format (*e.g.*, RGB, Postscript, TIFF, GIF, YUV). Another example is integrating profiling data [28] with protein-protein network interaction.

In J3DV, data importing is carried out by integrating data sources with the visualization tool. Considering that data sources may be of a wide variety, we choose the Java Data Object (JDO) specification as the backbone of the integration, which is implemented in the data access module mentioned earlier. As proposed by the Java Data Object Expert Group, JDO has three goals. First, JDO defines contracts and responsibilities for various roles, which leads to standard connectivity to data sources. In addition, this will also enable a standard JDO implementation for a data source to be pluggable across multiple application servers. Second, JDO provides a transparent interface for applications and helps developers to store data without learning a new data access language for each new data source. Third, JDO makes it easy for application developers to use the Java programming model to model the application domain and

transparently retrieve and store data from various data sources (*e.g.*, relational databases, object databases, mainframe transaction processing systems and local file systems).

The J3DV tool will transform the imported data into Java objects. Behind the transformation, there exists a data type mapping, which maps each data type of the data source into a Java data type. At the same time, the data type mapping is able to keep data precision and avoid losing information.

3.2 Querying

The querying capability of a visualization system is closely related to the data model. A good visualization system must be able to process and optimize queries based on a particular data model.

Generally, querying is not a well-developed area of visualization, and its implementation is *ad-hoc* in different systems. For example, SAGE supports retrieving previously created graphics based on appearance and/or data contents. On the other hand, most tools support drill-in and fly-by operations, which have similar effects to queries.

For J3DV, the resulting image is a graph, which consists of nodes and edges. The J3DV supports some query functions by providing searching and extending functions. A user can search a node entity by entering its key, which is one of the attributes of the data model to identify the node. The user can use the extending function to display all the nodes, which have direct connections with the current node. J3DV also provides other interaction methods, such as Position Control, Rotation Control, Move and Delete. All these methods provide navigational controls and manipulation to the resulting graphs.

3.3 Distribution

Data visualization tools are mainly used by scientists and researchers who perform experiments that result in large quantities of data. These data often need to be distributed over a network. Most visualization systems provide support for distribution. For example, MineSet by Silicon Graphics Inc. follows the client/server execution model. The client and server may reside on different computers over a network. The client presents visual representations to the user. The server performs most of the computation, including retrieving, transforming data and analyzing data.

J3DV is also distributed. Not only can the client and server be distributed over a network, but also the data sources can be distributed over a network because JDBC supports remote data access. As far as the Protein-protein interaction mapping project is concerned, the J3DV running on computers of the Computer Science Department can access the data sources in the Genetic Department. In addition, a J3DV server can provide services to multiple clients, and a client can get services from multiple servers.

3.4 Heterogeneity

Heterogeneity issues are often closely related to distribution issues. This is because many distributed scenarios involve heterogeneous systems. Unix is a favorite platform for visualization tools due to its powerful computational capability and high-end graphics. Most tools can run on a Unix platform. Some of them only target Unix. For example, the OpenDX can only run on X Window and motif. Others can run in mixed environments. For example, the AVS/Express runs on both Unix and Windows platforms.

The J3DV can run on most major platforms, such as Unix, Linux and Windows. Clients and servers can be deployed on different platforms and still communicate with each other. This portability comes from two things. The first is its implementation language, Java, which is “write once, run anywhere”. The second is a well-defined client-server communication interface, which makes the communication totally independent from the platforms.

4. Data Modeling and Mapping

4.1 Data Modeling

In all visualization tools, data modeling is an important concept. By defining a data model, a visualization tool becomes general-purpose by supporting a wide variety of data that cross discipline boundaries. The implementation of a visualization tool solely depends on the data model instead of any particular application domain. A good data modeling approach can smooth the data transformation, and be applied in different application domains. The essential problem to be solved here is how to present the rich information in the data source to the end-user in a freedom-limited display.

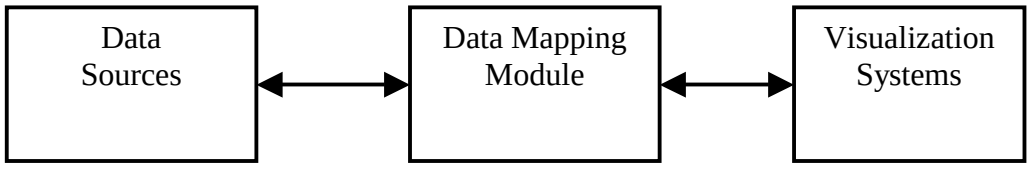

Figure 3. The Mismatch between Data Models of Database and Visualization.

Different data sources have different data models. For example, file systems may store data in columns and rows, where the data model could be a matrix. For a relational database, the data model consists of collections of relations. Each relation is a table. The columns of the table are atomic data types. The rows in the table are tuples of attributes.

Many visualization techniques, such as a 3D image, have been adopted to represent the data. These techniques use their own data models to define the visual representations. Visualization data models are usually data structure models with connectivity and topological specifications [19]. For example, MineSet uses the Tree Visualizer and Map Visualizer to model the data sets. As shown in figure 3, the data source with its underlying data model is one endpoint of the data visualization process. The other endpoint is a visualization system with a graphical data model. The two models abstract data differently, resulting in a mismatch between them. The critical issue for data visualization is how the data are transformed with an intermediate data model. Mapping is the method we use to bridge the two. The following section focuses on mapping.

4.2 Mapping

In essence, a mapping procedure is used to keep interesting information and filter out trivial or unwanted information, and transform the interesting information into the desired form. As Hibbard *et al.* [10] pointed out, there is no general data model for all applications. Correspondingly, there is no one single method that can be applied to all mapping situations. Nevertheless, many methods have been proposed and studied. For example, Haber *et al.* [8] proposed the fiber bundle data model; its main idea is that the input data of the scientific visualization are often sampled at grid points. Kao *et al.* [14] proposed the extended schema data model. This data model allows the user to store and manipulate scientific data in a uniform way.

The mapping procedure depends on the data model. In J3DV, we use a two-level mapping to transform data from its original state to an intermediate form, which can be used to render images by the visualization tool.

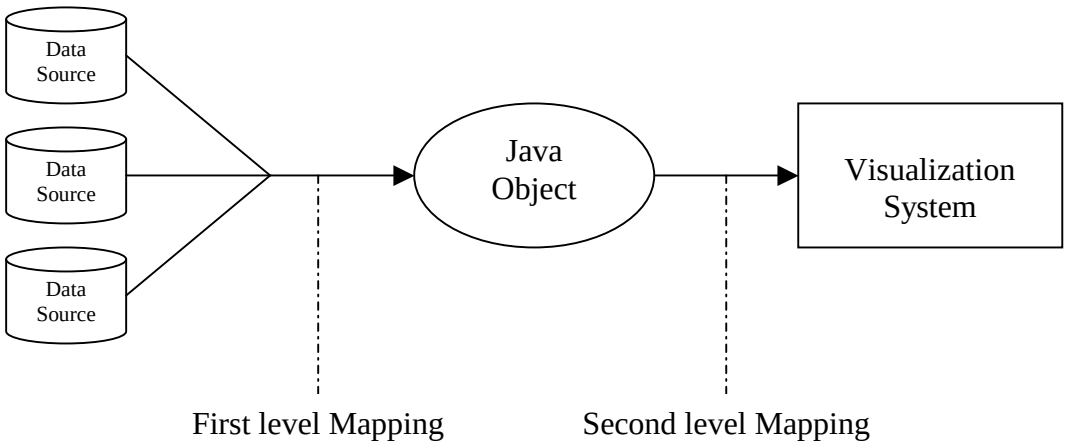

Figure 4. J3DV Mapping Procedure

In Figure 4, we present the mapping procedure. The first level mapping transforms original data in the data sources into Java objects. For each data source, there exists a data type mapping, which will transform the data from the data source into Java objects, *i.e.* group of attributes. After the first level mapping, data from different sources have a uniform structure, making it easier to process the data.

The second level mapping transforms the data in the form of Java objects into intermediate data. This mapping is based on the data model. The J3DV displays the data set in the form of graph. The data model of J3DV is therefore a graph data structure, which consists of entities of nodes and edges. A node is described by its shape, color and size, while an edge by a pair of nodes and their connection. In the Protein-Protein Interaction Project, we use nodes to represent proteins and edges to represent interactions between a pair of proteins. This data model can be applied to other application domains, where the data can be represented with a graph-like structure, such as network management and material science.

J3DV adopts a rule-based attribute-assembly mapping method to transform Java objects into the intermediate form. By rule-based, we mean that the mapping is done according to some user-specified rules. These rules generate predicates indicating the value of a given intermediate data attribute. By attribute-assembly, we mean that all the mapped data attributes will be assembled as an entity, that is, a node or an edge.

Using the two-level mapping, the intermediate data can be generated, and cached in memory and/or disk. The client can then access it via either JNDI or the RMI activation daemon.

5. Sample Run

In this section, we demonstrate how the J3DV tool can be used. [12] is the web page where user can download the source code of J3DV.

5.1 Server

Figure 4 shows the server-side's main window. The user can follow the specified order to create a new mapping. As is clear from the following window, it takes six steps to create a mapping.

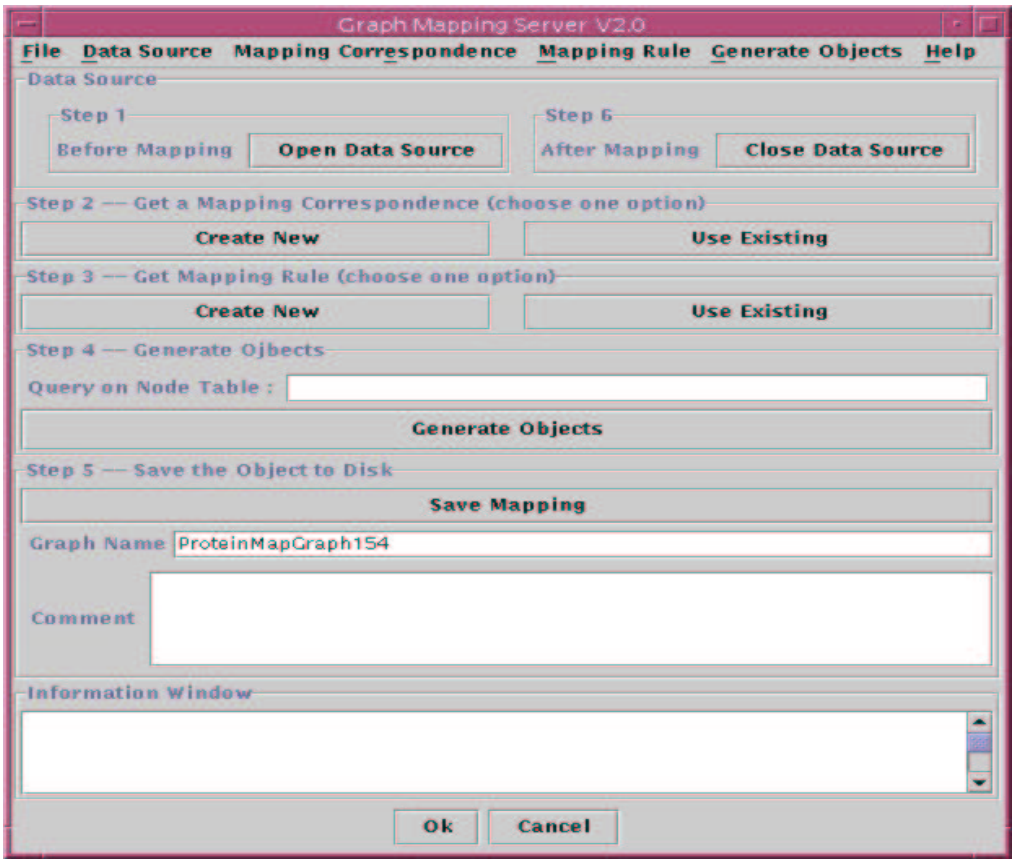

Figure 5. J3DV's Server-side User Interface

In step 1, the user needs to choose data sources. In step 2, the user specifies the data set in which he/she is interested, and the mapping correspondence between the

attributes of raw data and the attributes of data model. Figure 5 shows the schema of the database (such metadata can help a user create a mapping by providing attribute information). Figure 6 shows the mapping correspondence window, which helps the user to specify the mapping correspondence between the attributes of original data and the attributes of intermediate data model.

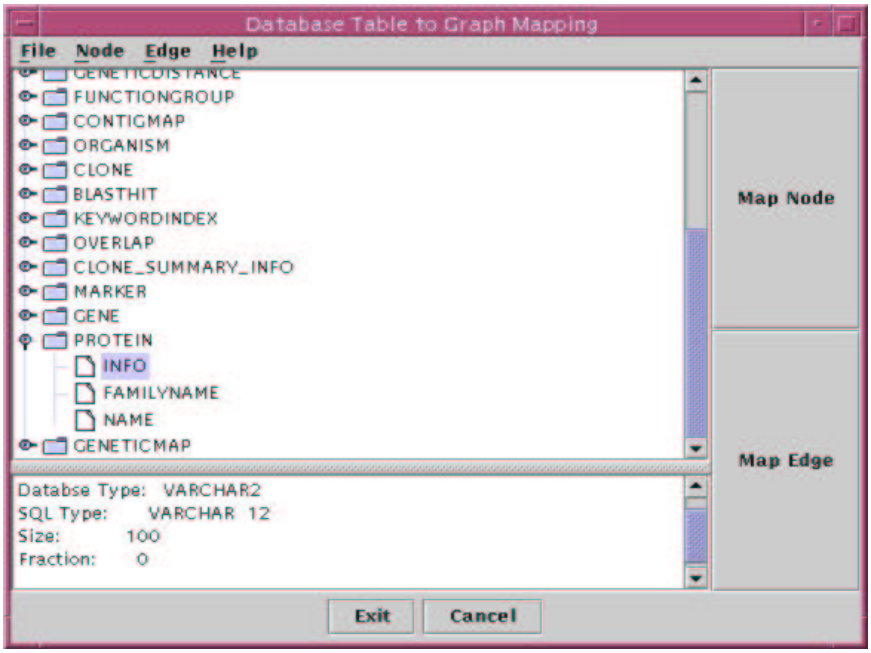

Figure 6. Visualized Database Metadata

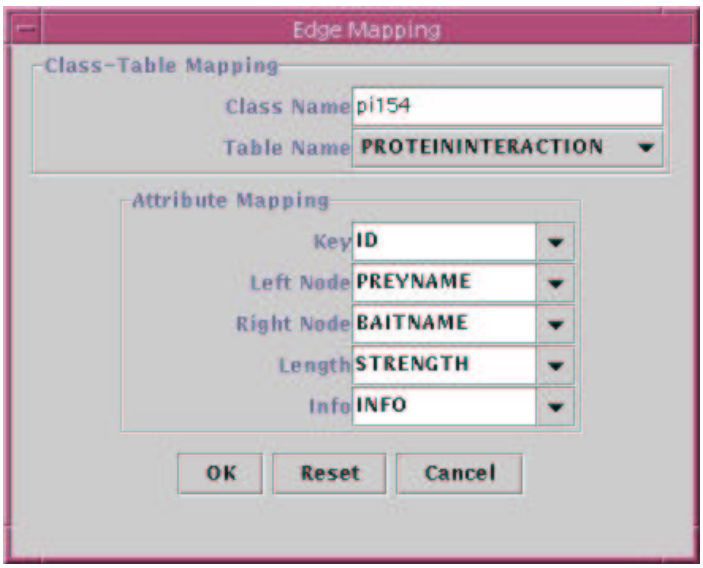

Figure 7. Mapping Correspondence

In step 3, the user needs to give the mapping rules. J3DV provides a mapping rule generator, as shown in Figure 7. Based on the mapping rules specified by the user, the rule generator will generate a Java class, whose function is to transform raw data into intermediate data.

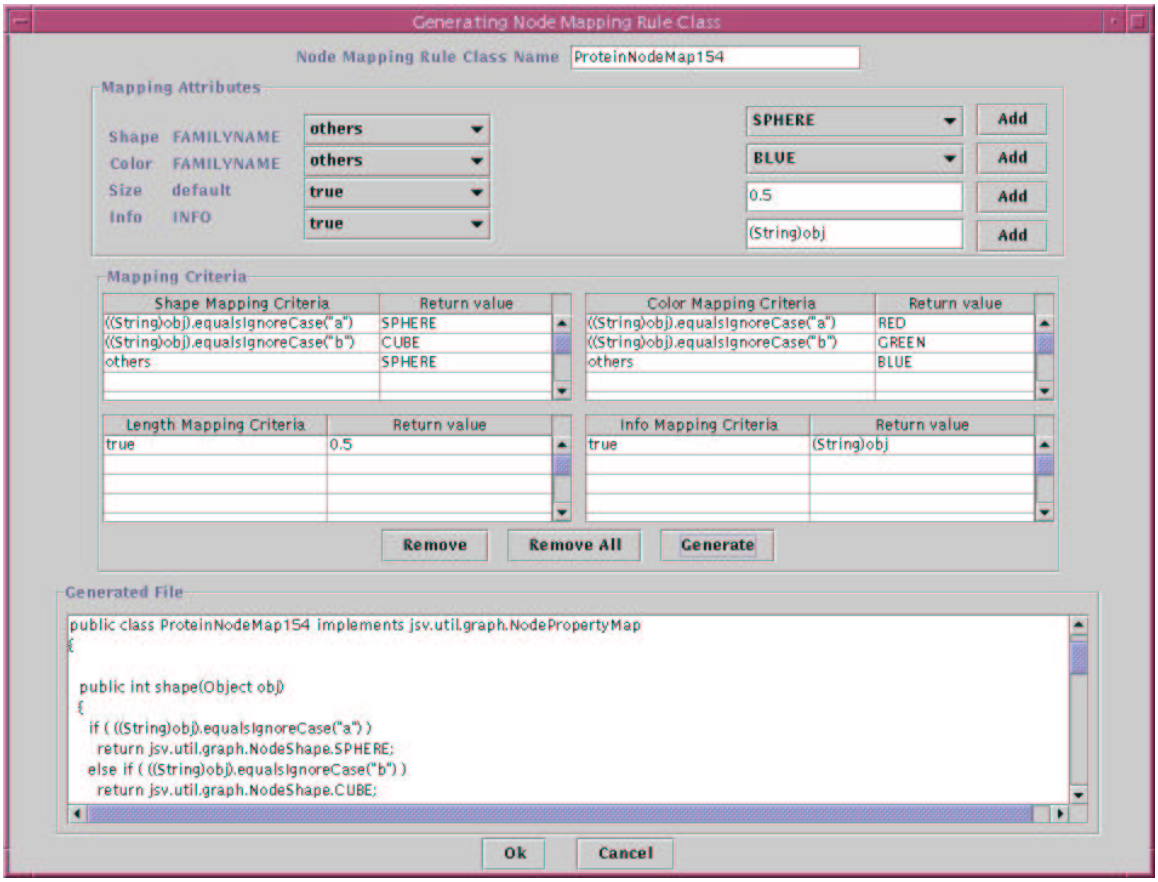

Figure 8. Mapping Rule Generator

In step 4, J3DV will actually transform the data into intermediate data based on mapping correspondence and mapping rules. Before that, J3DV will do error-checking to make sure that no fatal errors, such as missing mapping class, exist. In Step 5, J3DV saves the mapping information (mapping correspondence and mapping rules) for future reference, and caches the intermediate data into both caches. In Step 6, J3DV closes the connection to currently opened data sources.

5.2 Client-Side

Figure 8 shows the client's data request window. To use the server's services, a client has to know where the server process is, which is identified by a host name and a port number (default is 5002). The client can choose the map name, which identifies the intermediate data cached in the server. The server then sends the requested intermediate data to the client.

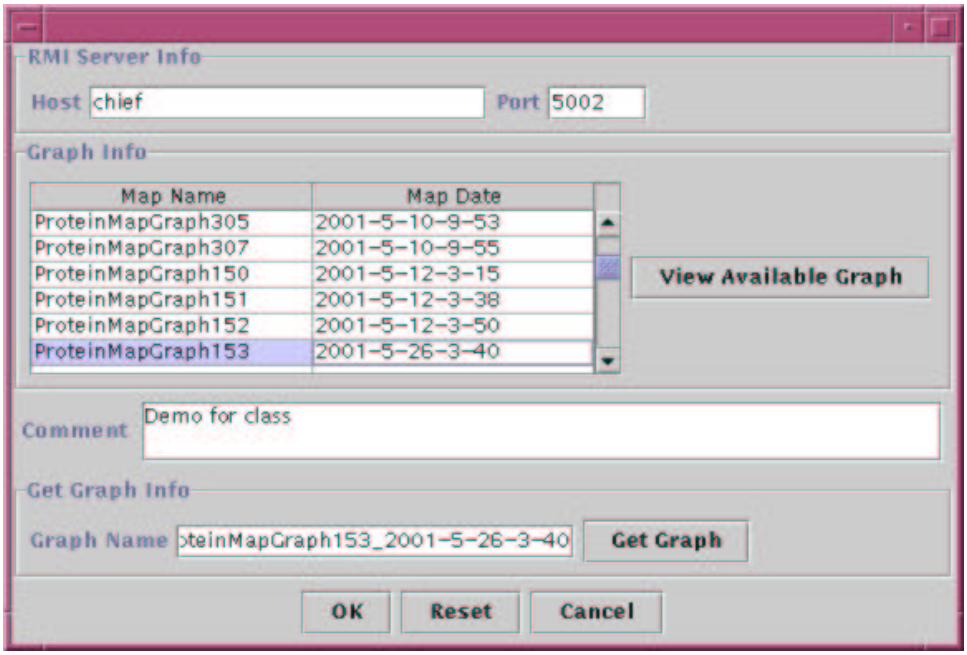

Figure 9. Client's Data Request Window

Once the client gets the data, the user can specify a layout algorithm from the menu. The J3DV provides several interaction methods. These include position control (which moves the graph along different axes), rotation control (which rotates the graph along the X, Y, and Z axes in the 3D space), extend (which extends the incident nodes and edges of the current node), move (which changes the position of a node in the layout graph), and delete (which removes a node from the layout graph). Figure 10 and 11 show the graph with initial layout algorithm and cluster layout algorithm applied respectively.

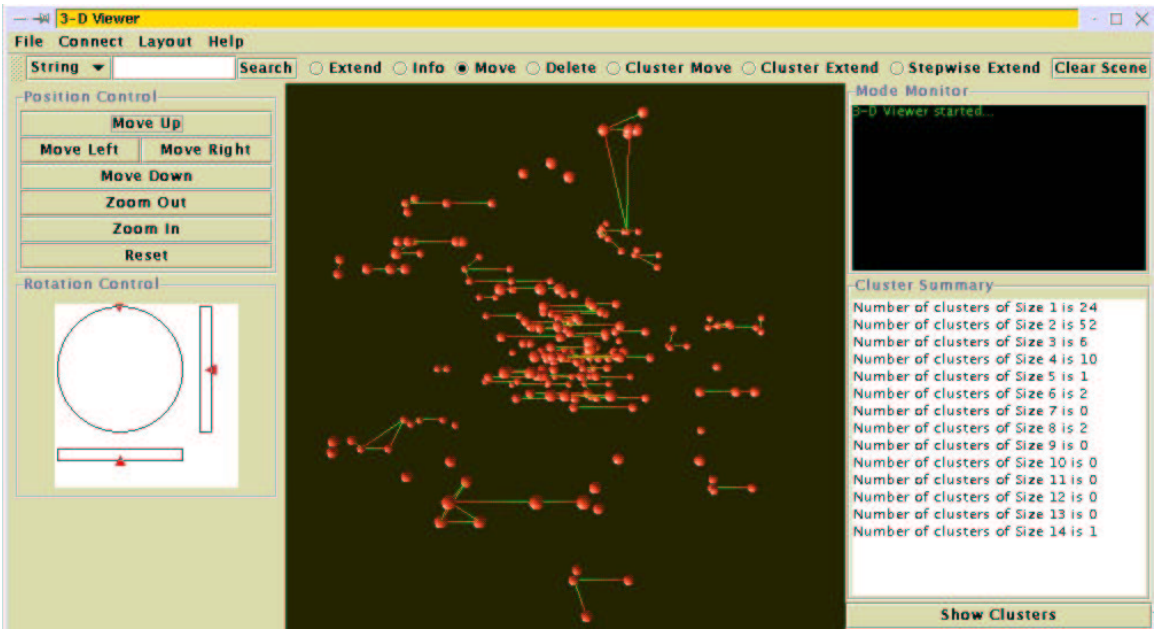

Figure 10. 3D Viewer displays the result of the initial layout algorithm.

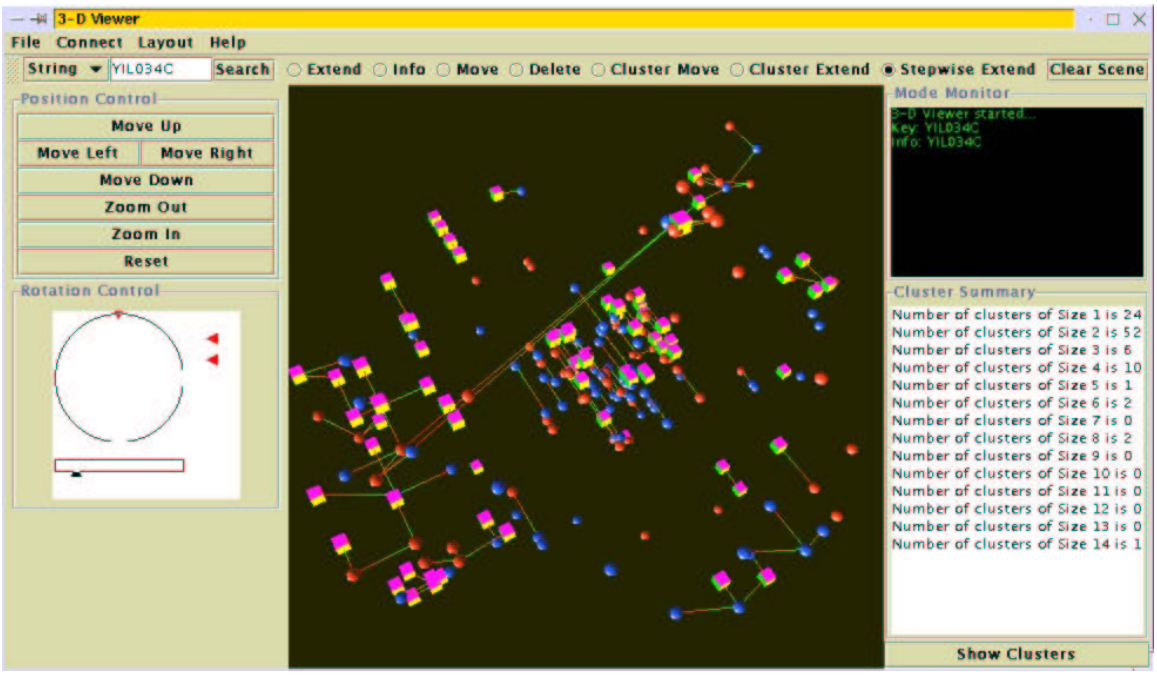

Figure 11. 3D Viewer displays the result of cluster layout algorithm

6. Conclusions and Future Work

In this paper, we introduced J3DV, a database visualization tool based on Java technology. It has been incorporated in the Protein-Protein Interaction Workflow developed at The University of Georgia. J3DV, together with other components of the workflow, such as ODS3 [31], helps researchers identify the protein interaction efficiently, which is helpful in understanding the functions of novel proteins.

The proposed system architecture has a sound framework. It differs from the currently available visualization tools in several aspects. The key feature of this system is that it uses the JDO specification to integrate different data sources with visualization. JDO allows efficient and uniform access to different data sources and provides a solution to the data management problem in database visualization. It is expected that many database systems will support JDO in the near future.

Extensibility is another feature that has been considered while designing the system. The extensibility is provided in three ways. First, we can easily plug in new data sources that implement the JDO specification. Second, we can plug in new data models to support more visual representations, such as transcriptional profiling information. Third, we can easily plug in a new graph layout algorithm.

In addition, J3DV utilizes a two-level mapping to smoothly transform the raw data into displayable data. It also utilizes a two-tier cache to provide better performance.

However, there are still some issues that need to be addressed in the future. The first is that we need to provide more visual representations, for example, a tree view, map view and histogram view. The second is that we need to integrate more types of data sources, such as object-oriented databases and XML. Third, we need to provide query

support for file systems. Fourth, in JDO implementation for file systems, we can use some standard metadata format to implement them, for example, Self-Defining Data Format [23] is such metadata format. Fifth, we need to provide an exporting mechanism, so that users can export the images as postscript [28], so they can be referenced later on without repeating the rendering procedure which is time-consuming for large data sets.

REFERENCES

1. Arya, M., N. Grady and G. Grinstein *et al.*, 1993, Database Issues for Data Visualization: System Integration Issues, Lecture Notes in Computer Science. Vol. 871, 1994, pp.16-24.

2. Asimov, D., 1985, The Grand Tour: A Tool for Viewing Multidimensional Data. SIAM Journal on Science Statistical Computing, Vol. 6, No.1, 1985, pp. 128-143.

3. AVS/Express, Advanced Visualization System, 2001, http://www.avs.com/products/ExpDev/expdev.htm

4. Baecker, R. M., 1986, An Application Overview of Program Visualization. Computer Graphics: SIGGRAPH 86, 20 (4), July 1986, pp.325

5. Beddow, J., 1990, Shape Coding for Multidimensional Data on a Microcomputer display. Proceedings of IEEE Conference of VISUALIZATION 90, pp. 238-246.

6. Berson, A. and S. Smith, 1997, Data Warehousing, Data Mining, and OLAP. Published by McGraw-Hill companies, 1997.

7. Goldstein, J. And S.F. Roth, 1994, Using Aggregation and Dynamic Queries for Expoloring Large Data Sets, Proceedings CHI'94 Human Factors in Computing Systems, ACM, Apr., 1994.

8. Haber, B., Lucas, B., and Collins, N., 1991, A Data Model for Scientific Visualization with Provisions for Regular and Irregular Grids. Proceedings of IEEE Visualization '91. San Diego, California. October, 1991.

9. Han, J. And M. Kamber, 2001, Data Mining: Concepts and Techniques, Morgan Kaufmann Publishers, 2001.

10. Hibbard W. Hibbard, Charles R. Dyer and Brian E. Paul: The VIS-AD Data Model: Integrating Metadata and Polymorphic Display with a Scientific Programming Language, Lecture Notes in Computer Science, Vol. 871, 1993. pp.37-68.

11. Ito, T., K., Tashiro, S. Muta, R. Ozawa, T. Chiba, M. Nishizawa, K. Yamamoto, S. Kuhara, and Y. Sakaki, 2000, Toward a Protein-Protein Interaction Map of the Budding Yeast: A Comprehensive System to Examine Two-hybrid Interactions in All Possible Combinations between the Yeast Proteins. PNAS, Vol. 97, no. 3, 2000, pp.1143-1147.

12. J3DV homepage, http://www.chief.cs.uga.edu/~jsim

13. JDO, Java Data Object Expert Group. Java Data Object. JSR000012, Version0.8. 2000. http://java.sun.com/aboutJava/communityprocess/review/jsr012/index.html

14. Kao, T., R. D. Bergeron, and T. M. Sparr, 1993, An Extended Schema Model for Scientific Data. Proceedings of IEEE Visualization ’93 Workshop, San Jose, CA, USA, Oct., 1993, pp.69-82.

15. Lang, U., G. Grinstein, R. D. Bergeron, 1995, Visualization Related Metadata, Proceedings of IEEE Visualization ’95 Workshop, Atlanta, Georgia, USA, Oct., 1995, pp.26-34.

16. Lee. J.P., 1993, Data Exploration Interactions and the ExBase System, Proceedings of IEEE Visualization ’93 Workshop, San Jose, CA, USA, Oct., 1993, pp.118-137.

17. Lee J. P., G. Grinstein, Lecture Notes in Computer Science: Database Issues for Data Visualization, 1993, Vol 871, pp. IX

18. MineSet, Silicon Graphics Inc., 2001, http://www.sgi.com/software/mineset

19. Nielson, G., P. Brunet, M. Gross, H. Hagen, S. Klimenko, Research Issue in Data Modeling for Scientific Visualization. In Scientific Visualization: Advances and Challenges. Academic Press, London, 1994, pages 472-479.

20. OpenDX, Open Data Visualization Explorer from IBM. 2000. http://www.research.ibm.com/dx

21. OpenGL, Silicon Graphics Inc., 2001, http://www.sgi.com/software/opengl

22. Owen, G.S., 1999, HyperVis --- Teaching Scientific Visualization Using Hypermedia, the ACM SIGGRAPH Education Committee, http://www.siggraph.org/education/materials/HyperVis/visgoals/visgoal0.htm

23. SDDF, 2001, Self-Defining Data Format, http://www-pablo.cs.uiuc.edu/project/SDDF/SDDFOverview.htm

24. Tian, H., 2001, Storage Management Issues for High Performance Database Visualization, *Proceedings of the 38th Annual Southeastern ACM Conference*, Athens, Georgia, Mar., 2001, pp. 251-256.

25. Visage, Carnegie Mellon University. 2001, http://www.cs.cmu.edu/~sage/sdm.html

26. Walhout, A. J. M., R. Sordella, X. Lu, J. L. Hartley, G. G. Temple, M. A. Brasch, N. Thierry-Mieg, M. Vidal, 2000, Protein Interaction Mapping in C. Elegans Using Proteins Involved in Vulval Development, Science, Vol 287, Jan., 2000, pp.116-122.

27. Wierse A., G.G. Grinstein, U. Lang Database Issues for Data Visualization, Lecture Notes in Computer Science, Vol 1183, 1995.

28. Wu, T., 2001, An Extensible Framework for Developing Visualization Software for Gene Expression Data, Master thesis, Department of Computer Science, the University of Georgia, Athens, GA, USA. Aug., 2001.

29. Zhang, Y. A Visualization System For Protein Interaction Mapping Using Java 3d Technology, Master thesis, Department of Computer Science, the University of Georgia, Athens, GA, USA. May, 2001.

30. Xu, Z., Mapping by Sequencing Using ODS3, Master thesis, Department of Computer Science, the University of Georgia, Athens, GA, USA. Jul., 2001.