GUS SB - A SCHEMA BROWSER FOR THE GENOMICS UNIFIED SCHEMA (GUS)

by

CONRAD V. IBAÑEZ

(Under the Direction of Eileen T. Kraemer)

ABSTRACT

The Genomics Unified Schema (GUS) is a relational database schema that supports a wide range of data types including genomics, gene expression, transcript assemblies, proteomics, and many others. Together with the GUS Application Framework and the GUS Web Development Kit (WDK), GUS is used by many genomics projects and groups to store diverse data and provide query-based websites for the scientific community. As the number of bioinformatics projects and groups using GUS as their schema of choice for storing information in their respective fields of study continues to grow, there is a need for an application allowing users to effectively browse information about GUS tables, especially relationships among them. Not only will such an application help current users understand the schema, but it will facilitate the popularity and use of GUS overall in genomics research. Thus, we introduce GUS SB, a schema browser and relationship viewer for GUS.

INDEX WORDS: Genomics Unified Schema, GUS, Schema Browser, Database Visualization, Information Visualization

GUS SB - A SCHEMA BROWSER FOR THE GENOMICS UNIFIED SCHEMA (GUS)

by

CONRAD V. IBAÑEZ

BBA, Georgia College & State University, 2003

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

MASTER OF SCIENCE

ATHENS, GEORGIA

2009

GUS SB - A SCHEMA BROWSER FOR THE GENOMICS UNIFIED SCHEMA (GUS)

by

CONRAD V. IBAÑEZ

Major Professor:	Eileen T. Kraemer
Committee:	John A. Miller
	Maria Hybinette

Electronic Version Approved:

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

DEDICATION

To my Lord and Savior, Jesus Christ.

ACKNOWLEDGEMENTS

First, I would like to thank God for all His blessings. I would like to thank my parents, sisters, and relatives both here in the U.S. and abroad who have given me support and prayers over the years. I would like to thank Dr. Kraemer for her tremendous patience, support, and guidance in this endeavor. I would like to thank the other professors on my committee, Dr. Miller and Dr. Hybinette, as well as other professors at the University of Georgia for helping me advance in this field of study and developing me into the person I am today.

I would like to thank all those who have helped me with this project. I would like to thank members of the Kissinger Lab at UGA, Dr. Kissinger, and especially Mark Heiges and Cristina Aurrecoechea. I would like to thank current and past members at the Computational Biology and Informatics Laboratory (CBIL), University of Pennsylvania Center for Bioinformatics, especially Steve Fischer for his input and Michael Saffitz who laid the technical foundation for this project. I would like to thank Joseph Hohenstern for providing the graphical images for GUS SB and for his presence along this journey. I want to thank the creators of open source software that have been utilized in GUS SB, especially the JUNG graph framework.

I would like to thank the Filipino-American Student Association at UGA and the Filipino community for making my years in Athens memorable. I would like to thank the graduate students who I met and have inspired me to complete my own work, especially Maria Mosqueda for believing in me. Finally, I would like to thank all the friends, especially from UGA, who have made an impression in my life.

TABLE OF CONTENTS

Page

LIST OF FIGURES

Page

CHAPTER 1

INTRODUCTION

The Genomics Unified Schema (GUS) is a relational database schema that supports a wide range of data types that include genomics, gene expression, transcript assemblies, proteomics and many others [1]. Together with the GUS Application Framework and the GUS Web Development Kit (WDK), GUS forms an extensible system for storing, integrating, and analyzing functional genomic data, which is the analysis of gene, RNA, and protein information and its biological function. GUS consists of seven schemas. Among them are DoTS (Database of Transcribed Sequences), RAD (RNA Abundance Database), TESS (Transcription Element Search System), SRes (Shared Resources), and Core, used for non-biological tracking and overhead. It has over four hundred tables. As the number of bioinformatics projects and groups that use GUS as the schema of choice for storing information in their respective fields of study continues to grow, there is a stronger need for an application that allows users to effectively browse tables in GUS, including all of their attributes and relationships. Thus, we introduce GUS SB, a schema browser and relationship viewer for the Genomics Unified Schema.

GUS is open source and is used in a number of projects, including GeneDB, TcruziDB, CryptoDB, ApiDB, PlasmoDB, BiowebDB, and many others listed at the GUS website [1]. Numerous genomics databases exist today such as Ensemble [2] and GMOD, the Generic Model Organism Database project [3]. To allow users to explore the use of GUS over other comparable genomics database schemas, users must be able to evaluate the schemas and determine which, if any, is suitable and is the best option to use for their research area. However, navigating and

comprehending large database schemas have proven to be difficult. Fortunately, certain tools exist or can be developed to help users browse large database schemas.

A well-designed schema browser for GUS will make it easier for users to explore and become familiar with the schema. An effective schema browser will help application programmers, especially those who lack background in biology and genomics, learn the GUS schema. It will show them table information from which they can determine where certain data should be stored. This will make development of applications built on the GUS framework become easier and faster.

Some projects may customize GUS to include new tables for storing data unique to their projects. Being that the GUS community is active, the GUS schema can evolve to include new tables per community needs and requests. The GUS Schema Browser can facilitate the evolution of the GUS schema through its use of visualization techniques to display relationships among tables. GUS SB will help contributors determine whether changes, both additions and modifications to the schema, are necessary.

GUS already has an online version of a schema browser that is available at its host site, which we will refer to as the *Online GUS Schema Browser*. The Online GUS Schema Browser was created by a team at the Computational Biology and Informatics Laboratory (CBIL) at the University of Pennsylvania. It uses Spring [4] and Hibernate [5] technologies and is powered by a database on the back end to store account-protected table and column documentation. The current web-based version of the GUS Schema Browser offers two basic views: a table-centric view and a category-centric view. The *table-centric view* displays the GUS tables grouped by their schemas as shown in Figure 1.1. The *category-centric view* groups the GUS tables by their super category and category as shown in Figure 1.2.

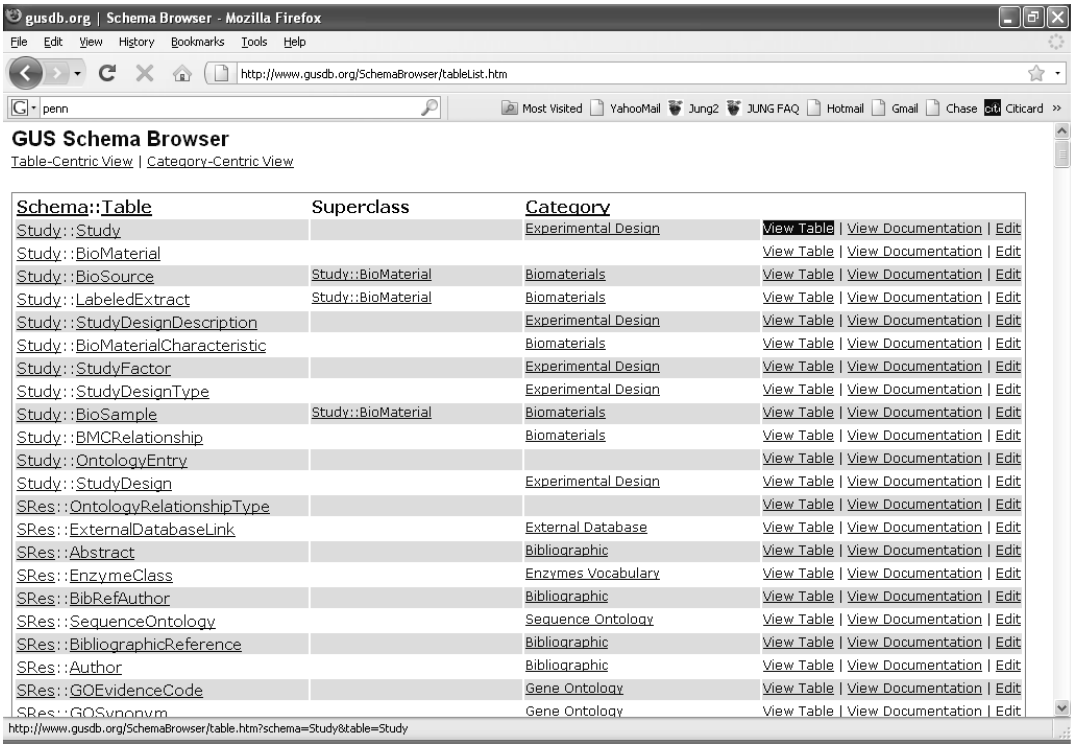

Figure 1.1: Online GUS Schema Browser – Table-Centric View

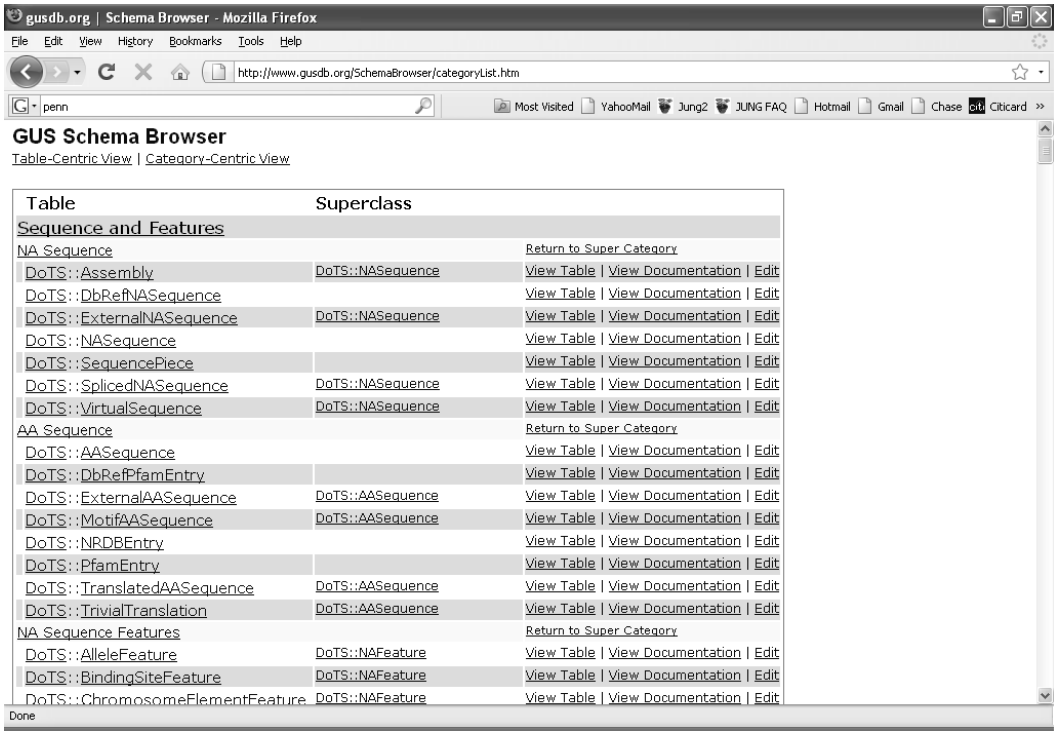

Figure 1.2: Online GUS Schema Browser – Category-Centric View

When a single table is selected in the current online GUS Schema Browser, all information for that table is shown to the user as can be seen in Figure 1.3. The displayed information includes the schema and table name, all of the columns for the tables, column types, description, and whether the column can be null. Child tables organized by category along with subclasses are also shown for the table.

We refer to *child tables* as those having a foreign key attribute that is the primary key of a particular table, called the parent. Therefore, *parent tables* are those tables containing the primary keys that correspond to the foreign keys found in the child tables. GUS also supports one-level subclassing as described in its documentation.

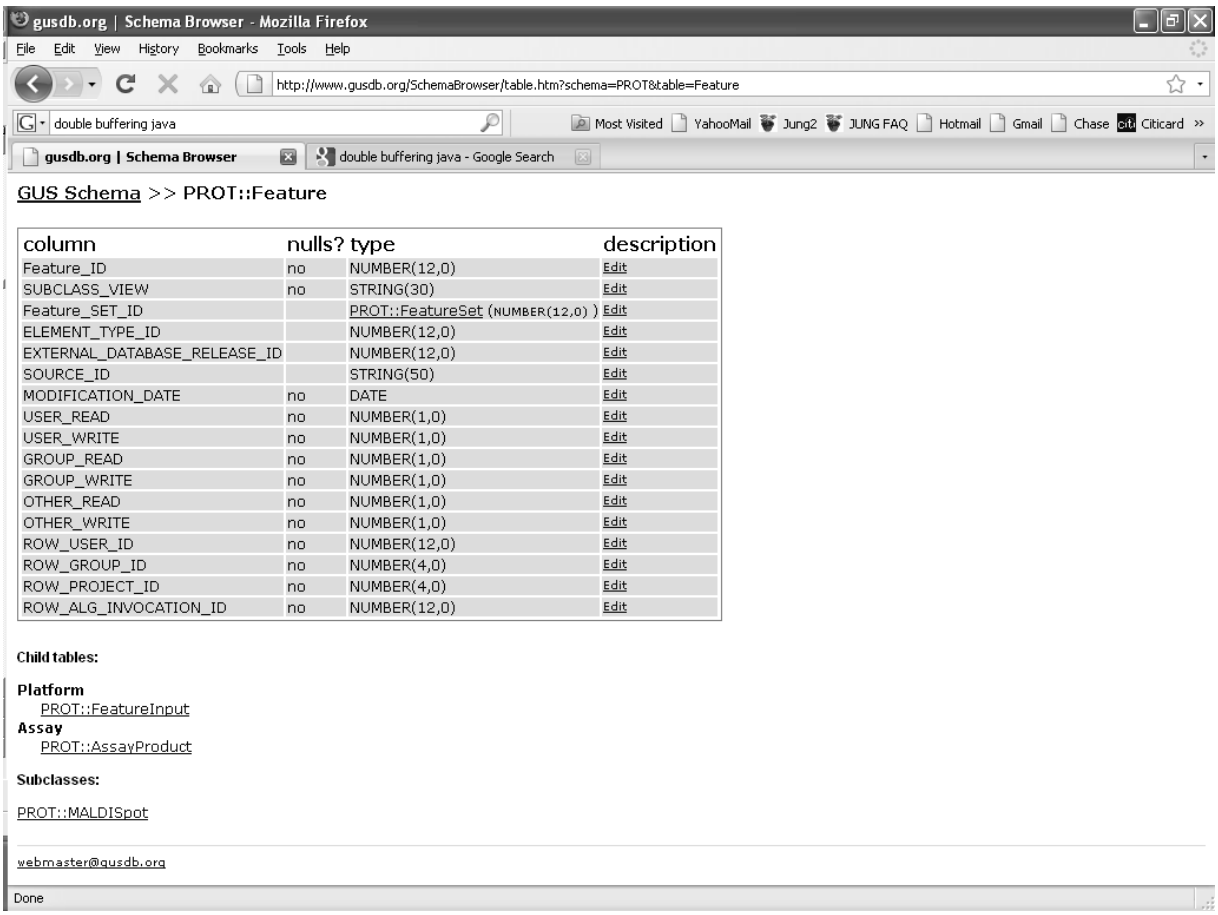

Figure 1.3: Online GUS Schema Browser Displaying the PROT::Feature Table

Numerous improvements in how the Online GUS Schema Browser displays information about the schema can be made. Currently, it is very limited in the visualization techniques it utilizes. It either shows a listing of all the tables without showing much information about each table, or it shows a single table and all of its information. Few HTML links and the forward and back buttons of the web browser provide limited navigation capabilities. Furthermore, it lacks an effective way of displaying the relationships among the tables.

For this project, we develop a *stand-alone version* of the Online GUS Schema Browser called the *GUS Schema Browser* or *GUS SB*, which includes both a schema browser and a relationship viewer. This stand-alone version can be uploaded to the GUS website and be made available for download by any potential user who can then begin to evaluate the GUS schema. The stand-alone version will add more functionality compared to the current online web version, such as a search feature, better layout and navigation controls, and a relationship viewer. GUS SB will allow prospective users to browse GUS without the need to install the database locally, obtain access to an existing GUS instance and connect to it via a database client, or purchase and install additional software on their local machines. We also implement enhancements to the Online GUS Schema Browser similar to features found in the stand-alone version that will improve overall navigation and visualization of table information on the website.

CHAPTER 2

BACKGROUND AND RELATED WORK

The GUS Schema Browser relies upon prior work and concepts implemented in today's existing technologies. In section 2.1, we present data modeling tools and their importance in database design. In section 2.2, we discuss the usefulness of database development tools. Section 2.3 covers tools for ontology creation and visualization. In section 2.4, we describe research in information visualization and information interfaces that can be applied to GUS SB. In the last section, we will discuss a similar project for browsing genomics schemas, called Schemaball [6].

2.1 Data Modeling Tools

Perhaps the most important tools for database architects are data modeling tools. Gary Cemosek describes certain characteristics of projects for which data modeling may not be necessary [7]. Projects where the domain is well known or the solution is trivial may not warrant data modeling. In addition, data modeling may not be as useful in projects where very few people collaborate and the scope of future needs and ongoing maintenance are minimal. However, these conditions are not usually the case with large enterprise projects in which numerous people or groups are involved and the functionality of applications expands over time. Bert Scalzo describes modeling as a way to obtain data requirements and transform them into a dependable database structural design [8]. For most projects, data modeling is an essential part of the software development life cycle.

Data modeling tools help database designers create diagrams that provide a logical view of data. A well-known example is the entity-relationship model first proposed by Chen in 1976 [9]. Entity relationship diagrams (ERDs) help to identify data entities and their relationships, as well as attributes. ERDs facilitate the creation of Data Definition Language (DDL) in Structured Query Language (SQL) which can be executed to create tables based on entities and foreign keys that represent relationships. Models generated by these tools also assist application developers who must manipulate the information stored in the database. The models provide documentation of where data is stored and facilitate in the creation of SQL statements in the form of Data Manipulation Language (DML) that are used to modify table data.

There are many data modeling tools available today with varying features and price ranges. They provide functionalities similar to those originally implemented in Schemadesign, a window-based tool developed by Sun Microsystems that allows users to graphically create and display database schema [10]. Existing open source data modeling tools include StarUML [11], ArgoUML [12], and Dia [13]. Some popular examples of professional tools are Microsoft Visio [14], Rational Rose Modeler [15], and IBM Rational Software Modeler [16]. The tools generally use the Unified Modeling Language (UML) to express data models. UML is a specification maintained by the Object Management Group (OMG) consortium that defines a graphical language for visualizing, specifying, constructing, and documenting the artifacts of distributed object systems [17]. Figure 2.1 shows a UML diagram generated by the StarUML application in the form of an entity relationship diagram.

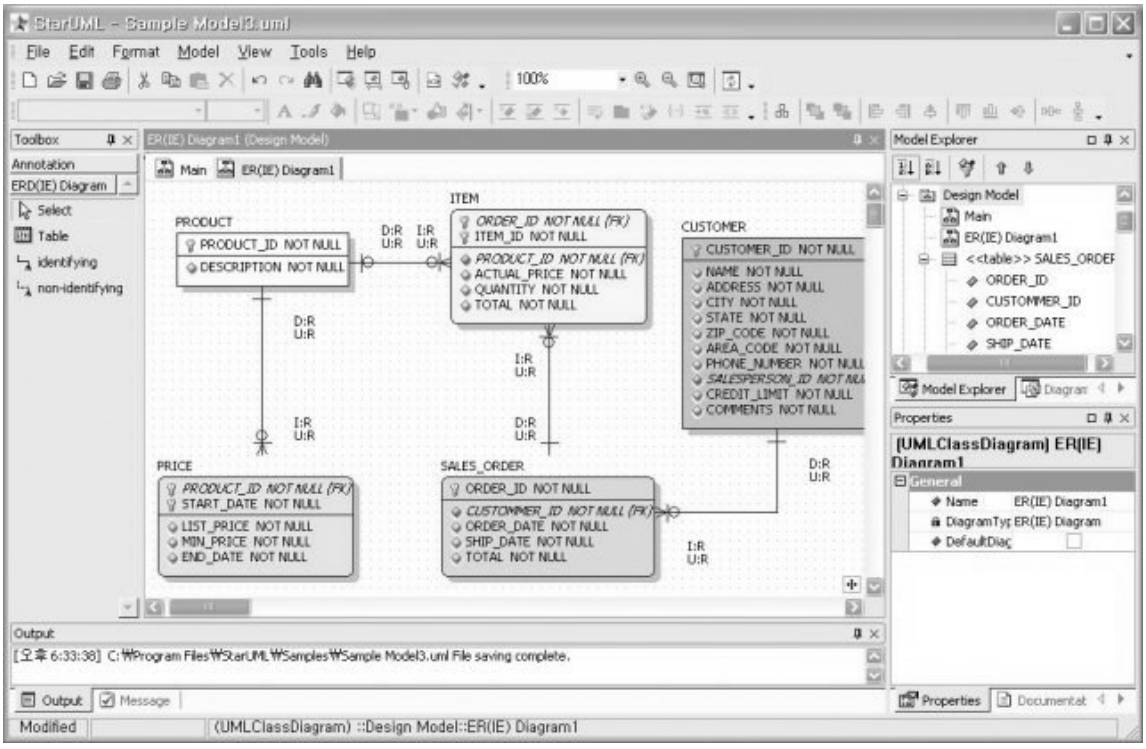

Figure 2.1: An ERD Diagram(taken from [11])

2.2 Database Development Tools

Numerous database development tools exist today, and most database management systems include such software as part of their package. Database development tools vary in features and price as well. Some open source applications are free to use while others are large-scale applications with abundant features and are priced in the hundreds or even thousands of dollars. Examples of open source tools are SQuirrel SQL Client [18] and PKLite SQL Client [19]. An example of a web-based open source data development tool is phpMyAdmin [20]. It is used for MySQL databases and is written in Hypertext Preprocessor language, PHP. Proprietary tools include PL/SQL Developer [21], Oracle SQL Developer for Oracle databases [22], and TOAD [23]. Most of these database development tools provide users the ability to perform

common tasks, which include the ability to browse the database and also make changes to the database schemas. They also allow users to manipulate underlying data through the execution of SQL statements. Figure 2.2 and Figure 2.3 show screenshots of how browsing and SQL execution are supported by the SQuirrel SQL client.

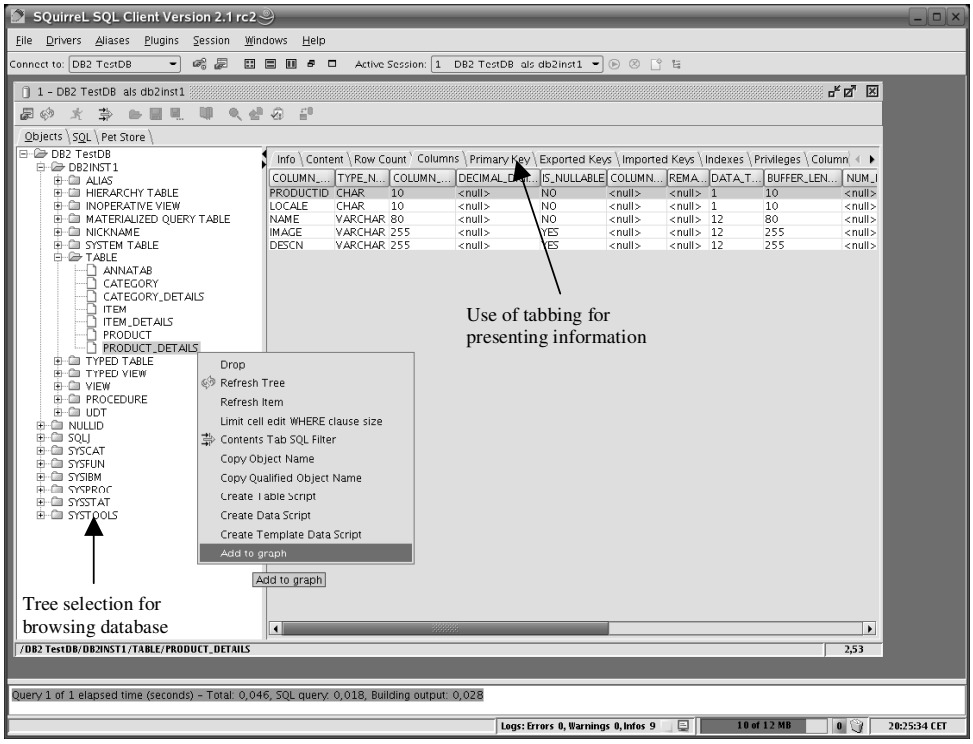

Figure 2.2: SQuirrel SQL Client – Schema Browsing(taken from [18])

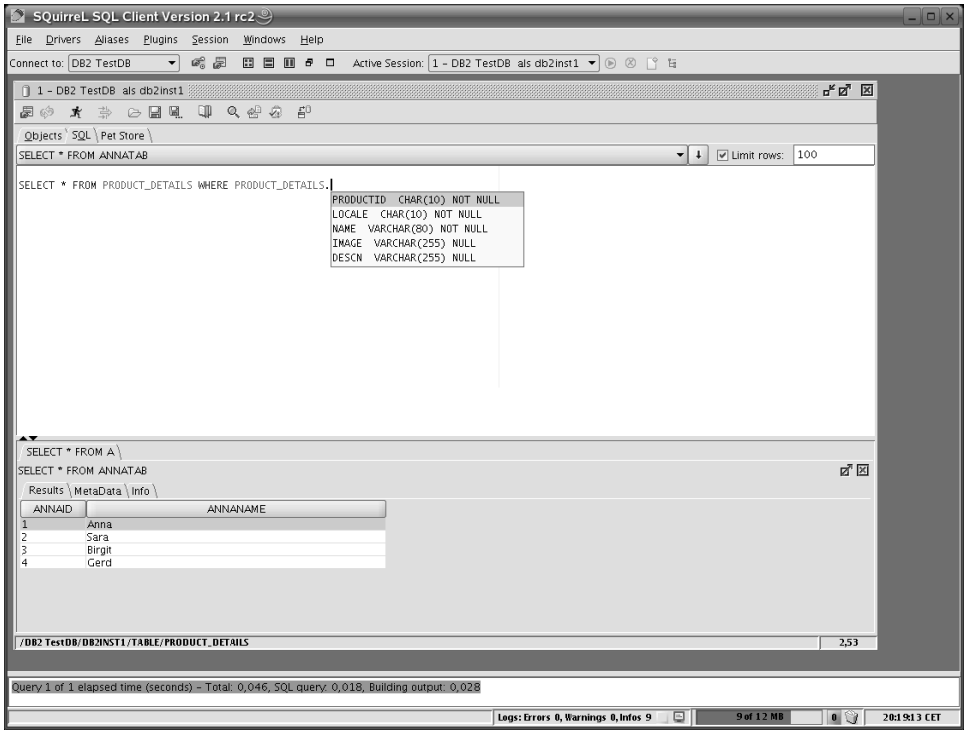

Figure 2.3: SQuirrel SQL Client - SQL Execution(taken from [18])

Current database development tools have not adapted some of the features described in prior work for browsing databases. The reason is that most of the database tools today focus on allowing the application programmer to develop complex SQL queries that when executed, return a result set or make changes to the data or schema. In the past, researchers proposed browsing tools that included an application layer in which users were able to browse and query the databases without knowing the underlying schema and without developing any SQL statements. Amihai Motro presented BAROQUE (BROwse and QUEry), a browser for relational databases. BAROQUE provides views that resemble semantic networks that combine both schema and data [24]. He stresses the importance of an alternative retrieval as opposed to systematic retrieval where users specify a formal query to satisfy their needs while the database management system retrieves the data for them. A related work is the "Bags and Viewers"

approach described by Inder and Stader, in which they propose interfaces that allow data to be retrieved without knowledge of database structures [25]. A bag is a logical container for a collection of objects that satisfy a set of constraints, and a viewer is attached to a bag that displays attributes of objects. Similarly, D'Atri et al. proposed ViewFinder, an object browser, which facilitates exploratory searches through the use of views attached to objects and frames, which hold object information such as members, superclasses, subclasses, and properties [26].

An interesting group of database development tools combine certain features of both database development tools and data modeling tools. This group includes the following proprietary software: Toad Data Modeler [27], DbVisualizer [28], CA Erwin Data Modeler [29], and DeZign for Databases [30]. Both the Toad Data Modeler and DbVisualizer are proprietary data modeling software that allow reverse engineering of databases by creating models of existing ones. DbVisualizer has most features associated with database development tools, while Toad Data Modeler can only change database schema through the creation of alter scripts which deploy any changes made to the data model into the corresponding database. CA Erwin Data Modeler and DeZign for Databases allow for both forward engineering and design generation, as well as database reverse engineering. One tool that is freely available and can be considered in this category of database development tools is SQL Developer [31]. SQL Developer provides users the ability to browse database structures and to create and execute SQL queries. It also has an integrated diagram editor that allows users to reverse engineer existing databases.

A closely related work to the GUS Schema Browser for visualizing database schemas is SchemaSpy. SchemaSpy is a Java-based tool that generates a web-based visual representation of a database schema [32]. It allows users to navigate through a hierarchy of tables using child and parent table relationships, which are represented by HTML links. It generates entity-relationship

diagrams of the schema. It also provides for a limited amount of interaction with users by allowing them to expand table relationships up to two degrees of separation.

2.3 Ontology Development Tools

T. R. Gruber defines an ontology as an explicit specification of a conceptualization, a description of the concepts and relationships that can exist for an agent or a community of agents [33]. An ontology contains a domain with objects and explicit relationships among them. As the use of ontologies continues to expand, so does the rise of development tools that are used to create and browse them, similar to those that exist for databases today. Protégé [34] is an open source platform of tools used in developing domain models and knowledge-based applications with ontologies. Noy et al. described Protégé as a conceptual modeling tool that can be used to express concepts and relationships in a domain [35]. Protégé-Frames editor and Protégé-OWL editor are used for building ontologies and provide interfaces that are similar to those found in database development tools. Furthermore, extensions of Protégé provide views comparable to data modeling tools as seen in Figure 2.4.

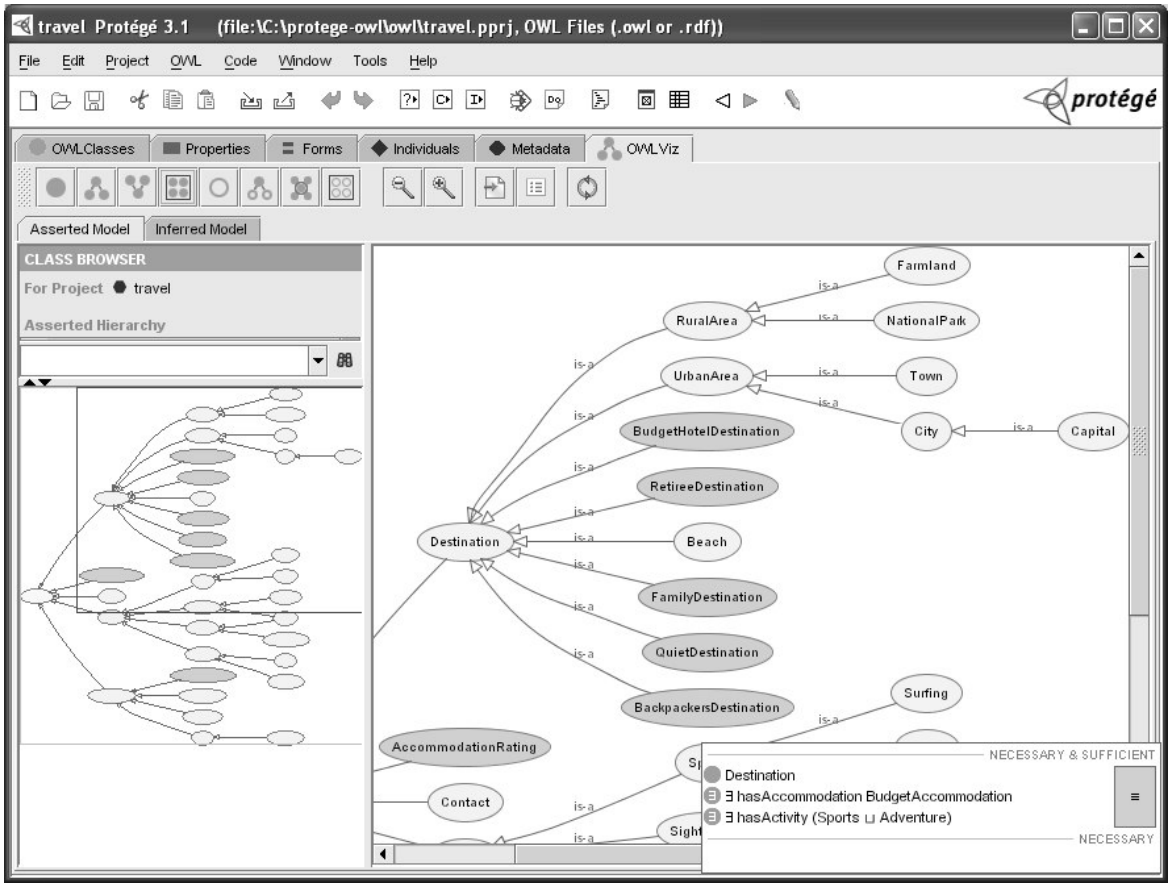

Figure 2.4: OWLViz Extension to Protégé(Taken from[34])

2.4 Information Visualization and Information Interfaces

GUS is a large relational database and contains close to five hundred tables. Because of this, we look into research that investigates different ways to display large amounts of information in a meaningful way. Thus, we briefly discuss research in the fields of information visualization and information interfaces.

There is much research in graph visualization and navigation in information visualization. Herman et al. discuss the issues that large graphs pose [36]. One issue is that large graphs can compromise performance and usability. Although displaying an entire large graph has the advantage of showing the overall structure and relationship between nodes, detailed analysis and

comprehension of the data in the graph become difficult. They also describe graph drawing problems that include aesthetic rules, such as minimizing edge crossings, minimizing the number of bends, and maximizing symmetry within graphs.

Displaying substantial amounts of information simultaneously to users can hinder their ability to distinguish between important and irrelevant information. This situation describes the current design and layout of the current Online GUS Schema Browser. Cockburn et al. describe several interface approaches that allow users to work at, and navigate between, focused and contextual views of a dataset [37]. We are especially interested in two interface schemes: zooming and cue-based techniques, which selectively highlight or suppress items within the information space. For the relationship viewer, we must consider the interaction controls for zooming in and out and also the relationship between zoom and controls for movement (panning). The concept of *semantic zoom* is also introduced in which objects are represented differently depending on different scales. The idea of cue-based techniques for rendering objects differently to gain focus is also applicable for GUS SB implementation.

The foundation of the designs for current modeling and database development tools can be described in the survey of Visual Query Systems (VQSs) discussed by Catarci et al. [38]. They define Visual Query Systems as systems for querying databases that use visual representations to depict domains of interest and related requests. They describe *browsing* as a viewing technique for gaining knowledge about the information content of a schema. They also distinguish among different types of users and the different types of VQSs that fit user needs. For example, they suggest that sophisticated users, who develop complex queries that require a good understanding of the database schema, may find diagrammatic systems useful for learning the data model.

2.5 Schemaball

A closely related project to the GUS Schema Browser is Schemaball, developed by Martin Krzywinski for the Ensembl project [2]. Ensemble is a project for producing genome databases for eukaryotic species. To address the issue of ERDs becoming difficult to follow for very large databases, Krzywinski proposes Schemaball as a circular method of composition for producing schema images [39]. Even with this method, comprehending relationships is still difficult in large graphs, so Schemaball has functionality for highlighting links and tables based on a recursive constraint trace. An example of an image generated by Schemaball can be seen in Figure 2.5.

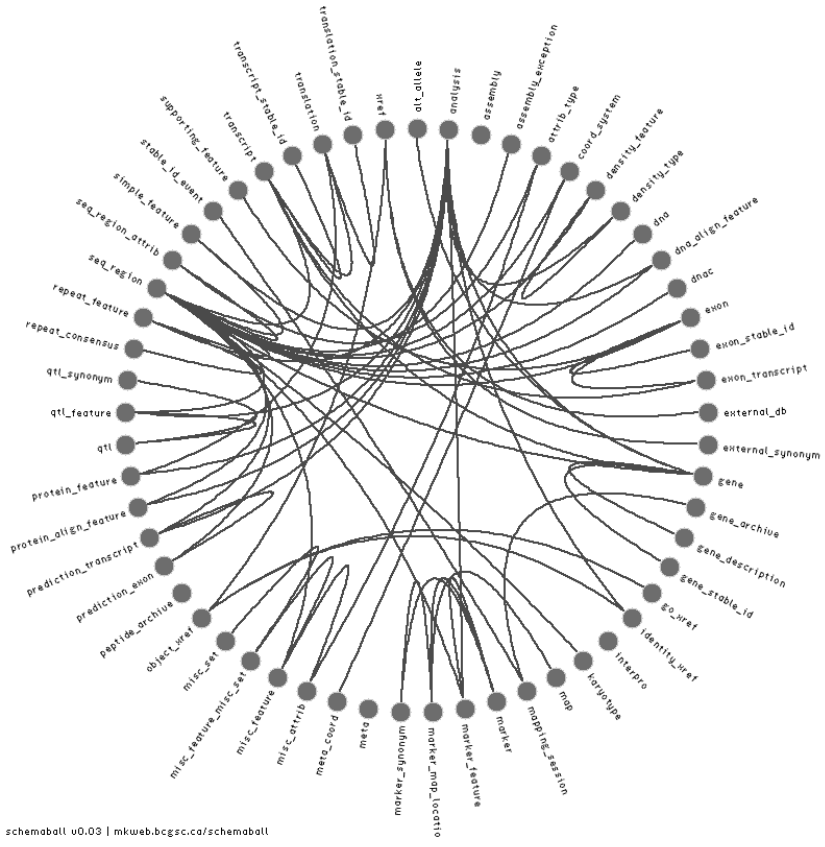

Figure 2.5: Schemaball Image for *Anopheles gambiae* Genome Database(taken from [6])

CHAPTER 3

GUS SB - SCHEMA BROWSER

This project applies concepts discussed in the background and related work section and focuses on improving the Online GUS Schema Browser. In the next two chapters, we discuss new features that are implemented in the stand-alone version, which we will refer to as the GUS Schema Browser or simply GUS SB. In this chapter, we focus on the schema browser module. Furthermore, we discuss improvements for the Online GUS Schema Browser that we have implemented.

3.1 GUS Schema Browser Basic Layout and Controls

The GUS Schema Browser provides many of the same functions as the Online GUS Schema Browser but attempts to present information to its users in a more effective design and layout. It essentially utilizes the same layout pattern that can be found in many of the existing database tools. As shown in Figure 3.1, the right content panel of GUS SB is devoted to displaying information. The left side contains controls that determine what should be displayed in the content window. A view options drop-down menu allows the user to toggle between a selection tree that organizes tables based on categories or schemas to which they belong. Backward and forward buttons allow the user to show previously displayed tables. A button with a magnifying glass image opens a search window, while a button containing a graph image initializes the Relationship Viewer.

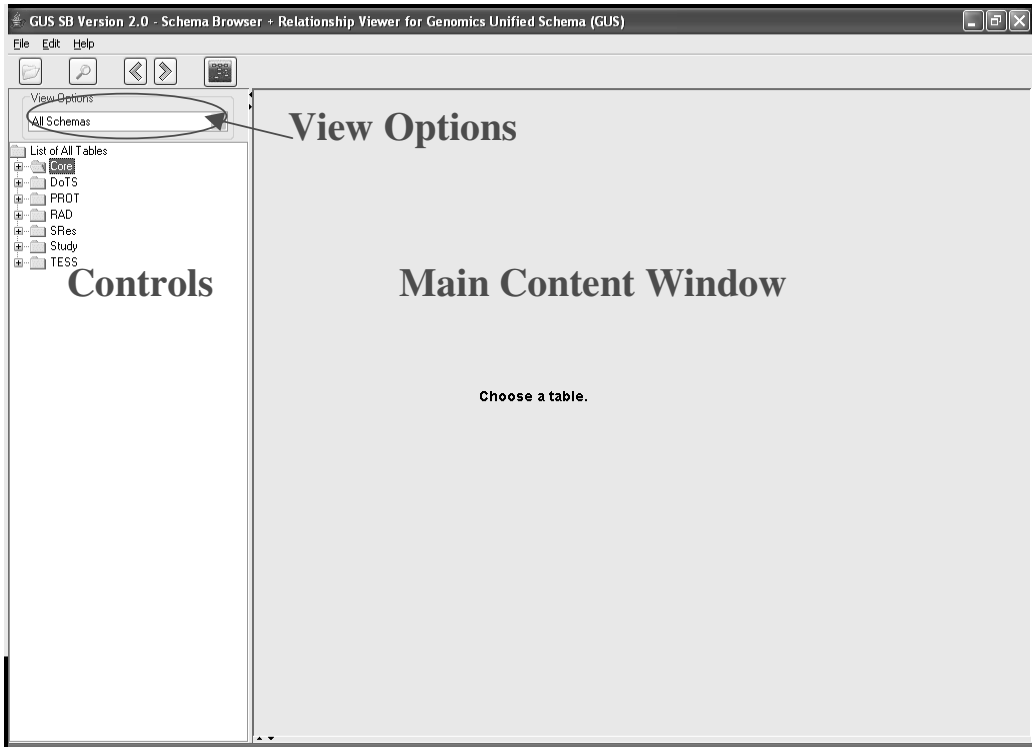

Figure 3.1: GUS Schema Browser Basic Layout

3.2 GUS Schema Browser – Table Browsing

Instead of displaying all table information in a single flat view, the GUS Schema Browser separates table information and hides data that is not in focus through the use of tabbed browsing. This limits the amount of information displayed to users. It allows them to control what information they can see and to concentrate on particular details of the table. We consider attributes as the most important characteristics of a table, and thus, that information is displayed in the main tab view, as seen in Figure 3.2. Other tab views include children, parent, and description. Figure 3.3 shows how the children tab view displays a list of child tables in a tree structure. The parent tab view displays parent tables in a similar tree structure.

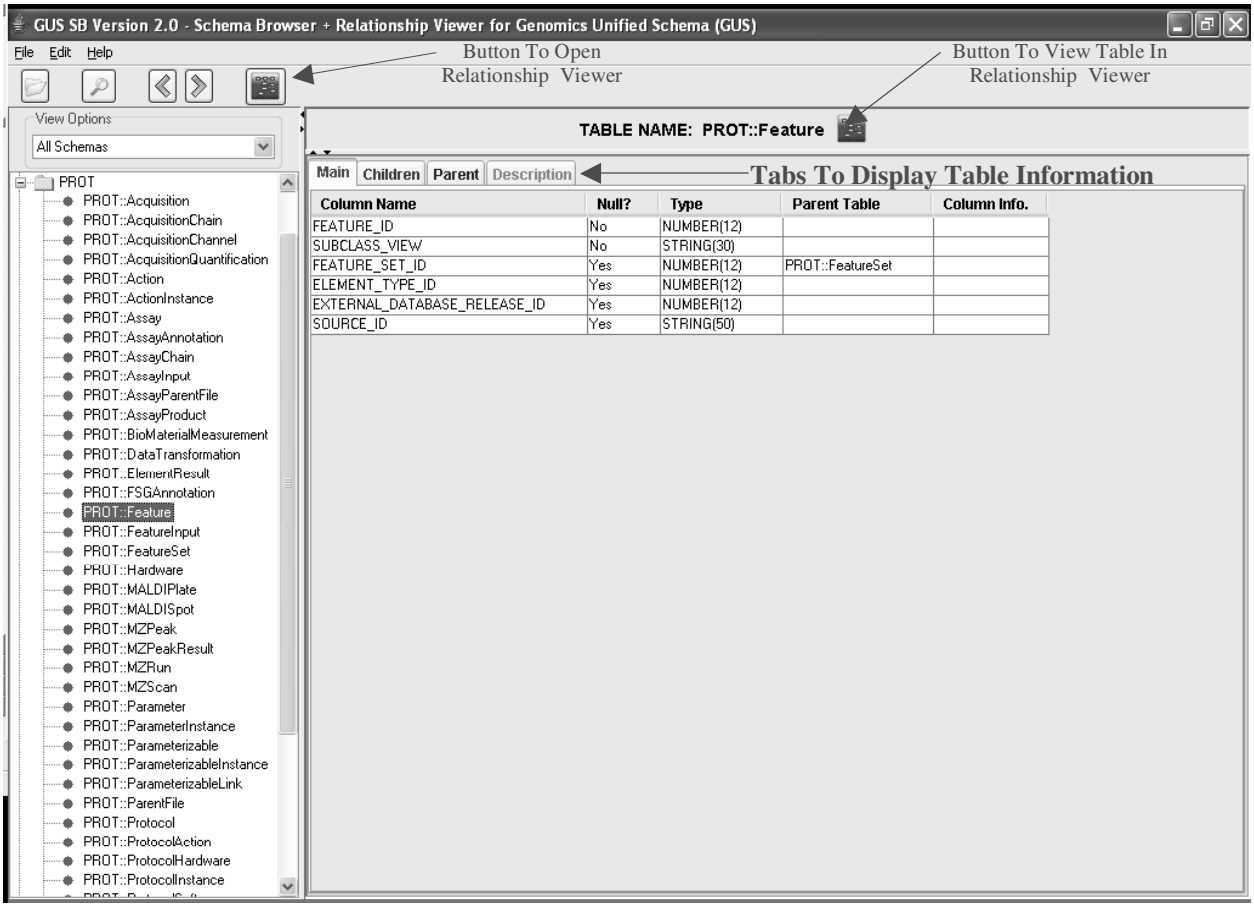

Figure 3.2: GUS Schema Browser Displaying Single Table

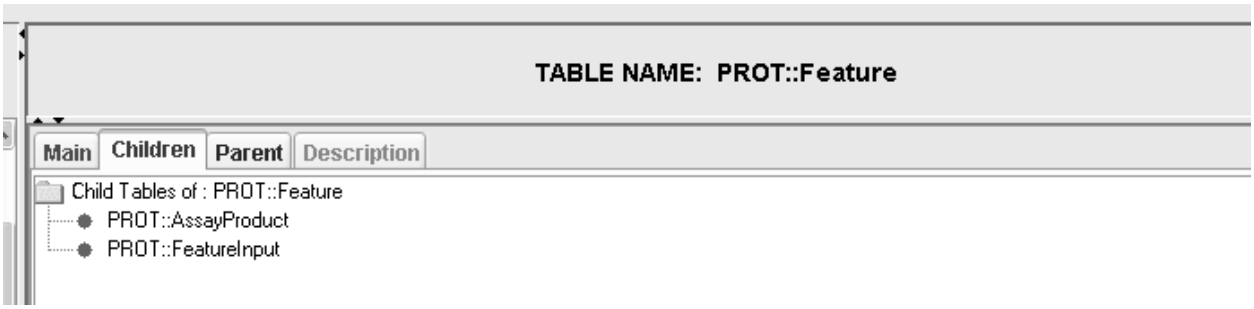

Figure 3.3: Children Tab Displaying Children Tables in a Tree

3.3 GUS Schema Browser – Find Keyword Feature

The GUS Schema Browser allows a user to search the schema for a keyword. It returns tables that contain the keyword in the options that the user specifies. Those search options are characteristics of the GUS tables and include Column Name, Column Info, Table Name, Parent, Child, or Table Info.

The Find Keyword feature is very useful. It helps users explore relationships by using the parent or child option. For example, the user can choose the 'Parent' option and enter the complete name of a specific table in GUS to use in the search. The search results will contain all tables that have that specific table as a parent. The search functionality also assists users in determining tables in which to store data by allowing them to search for a term in the column name and column information. To perform a similar search using a database development tool, users would need to connect to a GUS instance and then compose and execute SQL queries on database system tables that yield the same results. Figure 3.4 shows the Find window and list of options. Results are displayed in a box at the bottom of the Find window. Matching keywords in the table view are colored in blue.

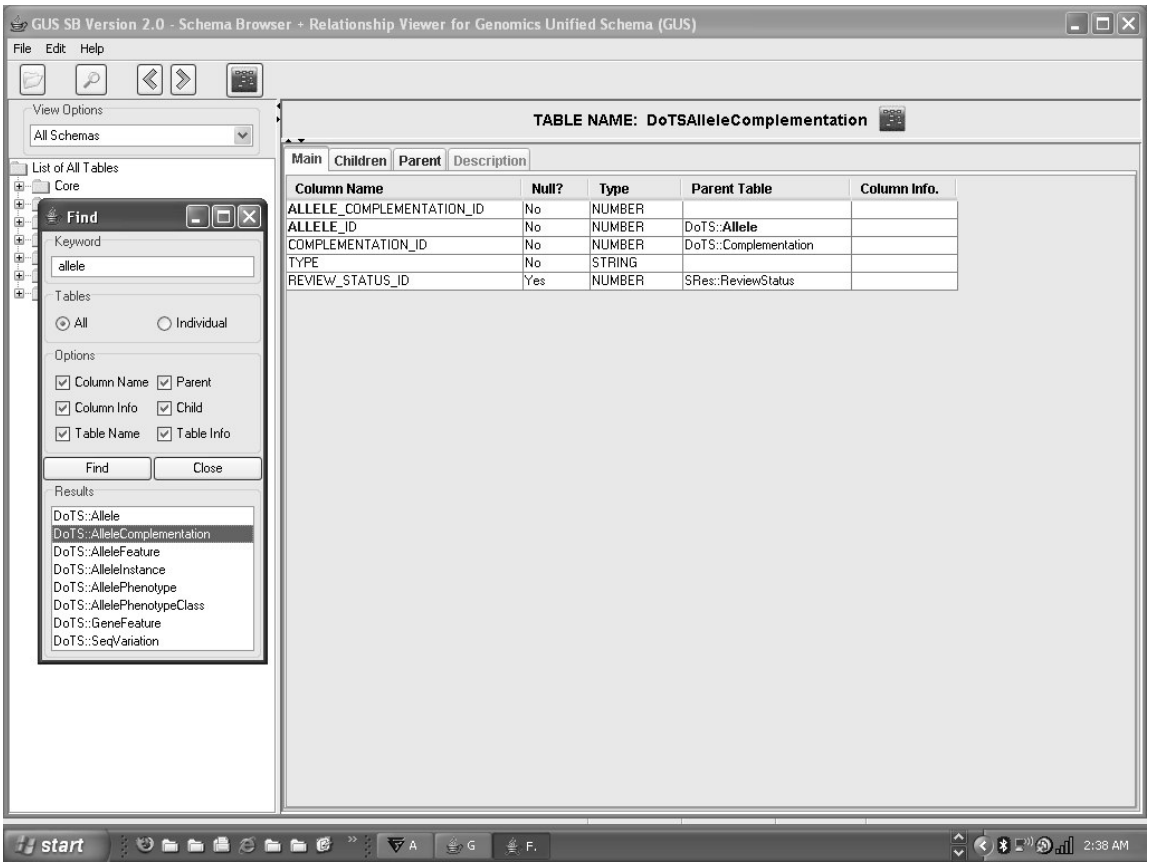

Figure 3.4: GUS Schema Browser Find Functionality

3.4 Online GUS Schema Browser Enhancements

Based on the features introduced in the stand-alone GUS Schema Browser, we have enhanced the Online GUS Schema Browser as seen in Figure 3.5. First, we implement a new layout similar to the one in GUS SB where the controls are on a left panel and the content window is to the right. We introduce tree structures for navigating between tables and tabbed browsing for focusing on specific table data.

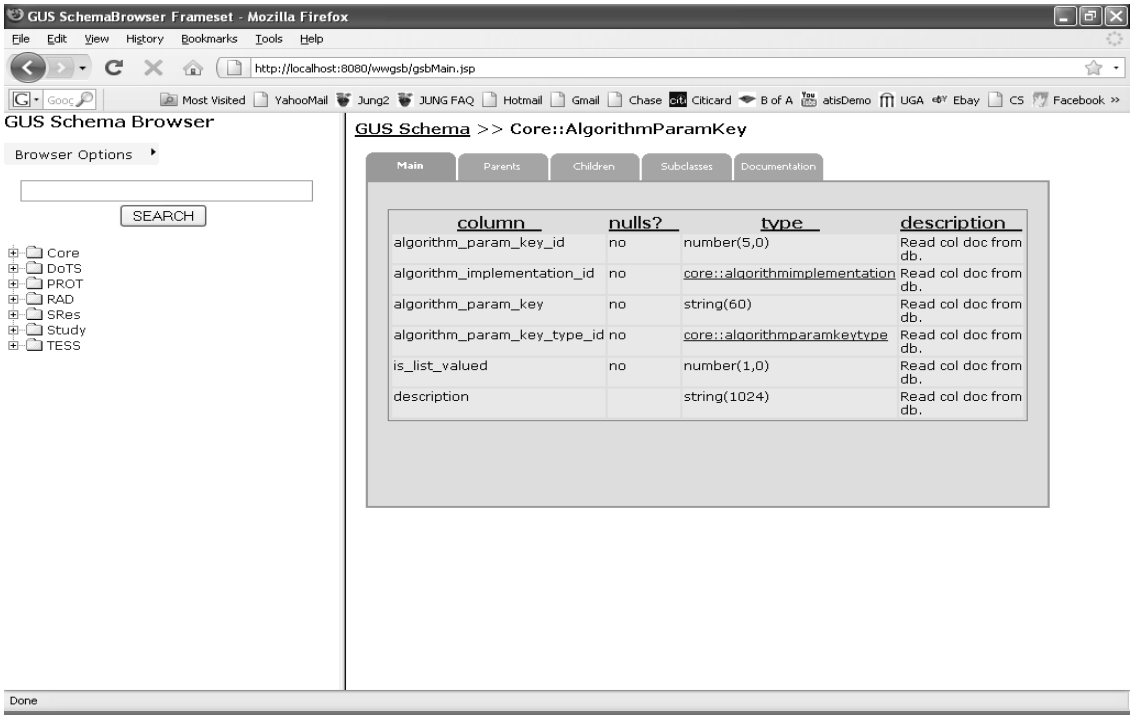

Figure 3.5: Layout Changes to Online GUS Schema Browser

A menu bar is also displayed to allow the users to choose between different views or different trees to navigate either by schema or category as shown in Figure 3.6 and 3.7 respectively. We introduce a similar search feature as was implemented for the stand-alone GUS SB. The basic search functionality returns tables that contain the search string in their schema name, table name, table info, column name, column info, parent, child, super category, or category information. Smart Search allows users to specify which of those table characteristics to apply in the search as seen in Figure 3.8.

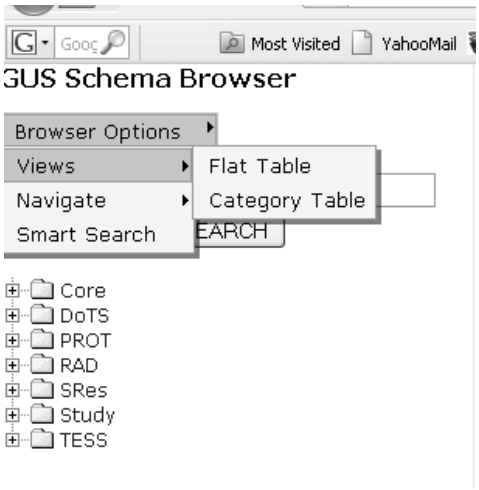

Figure 3.6: Drop-down Menu Views

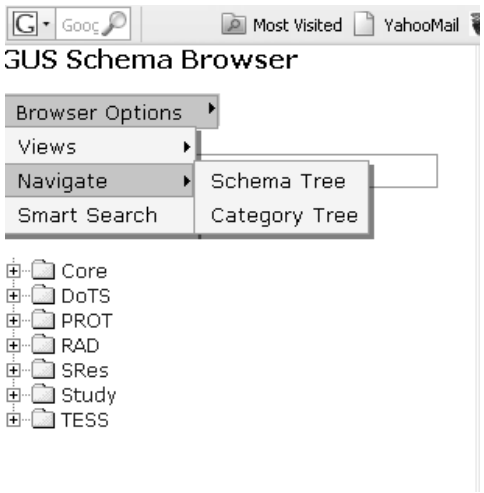

Figure 3.7: Drop-down Menu Navigate

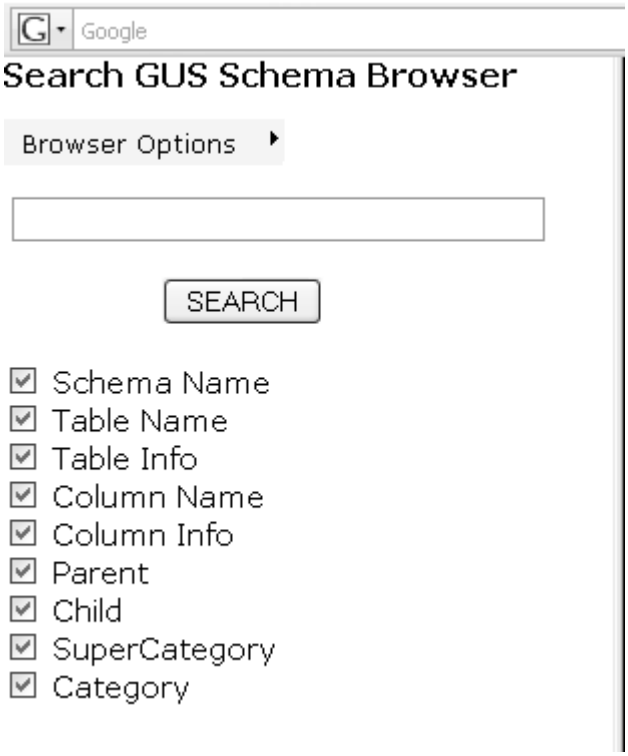

Figure 3.8: Online GUS Schema Browser Smart Search

To accomplish tabbed browsing for enhancements to the Online GUS Schema Browser, we use the Dojo Toolkit [40] that features DHTML and AJAX functions. We also use the toolkit

to implement a predictive text search functionality similar to those used by Google, YouTube, and other websites. Not only will this speed up the process in searching for keywords, but it will also broaden searches by proposing search terms in GUS that may be unknown to the user. Figure 3.9 shows the predictive text (type-ahead) search functionality.

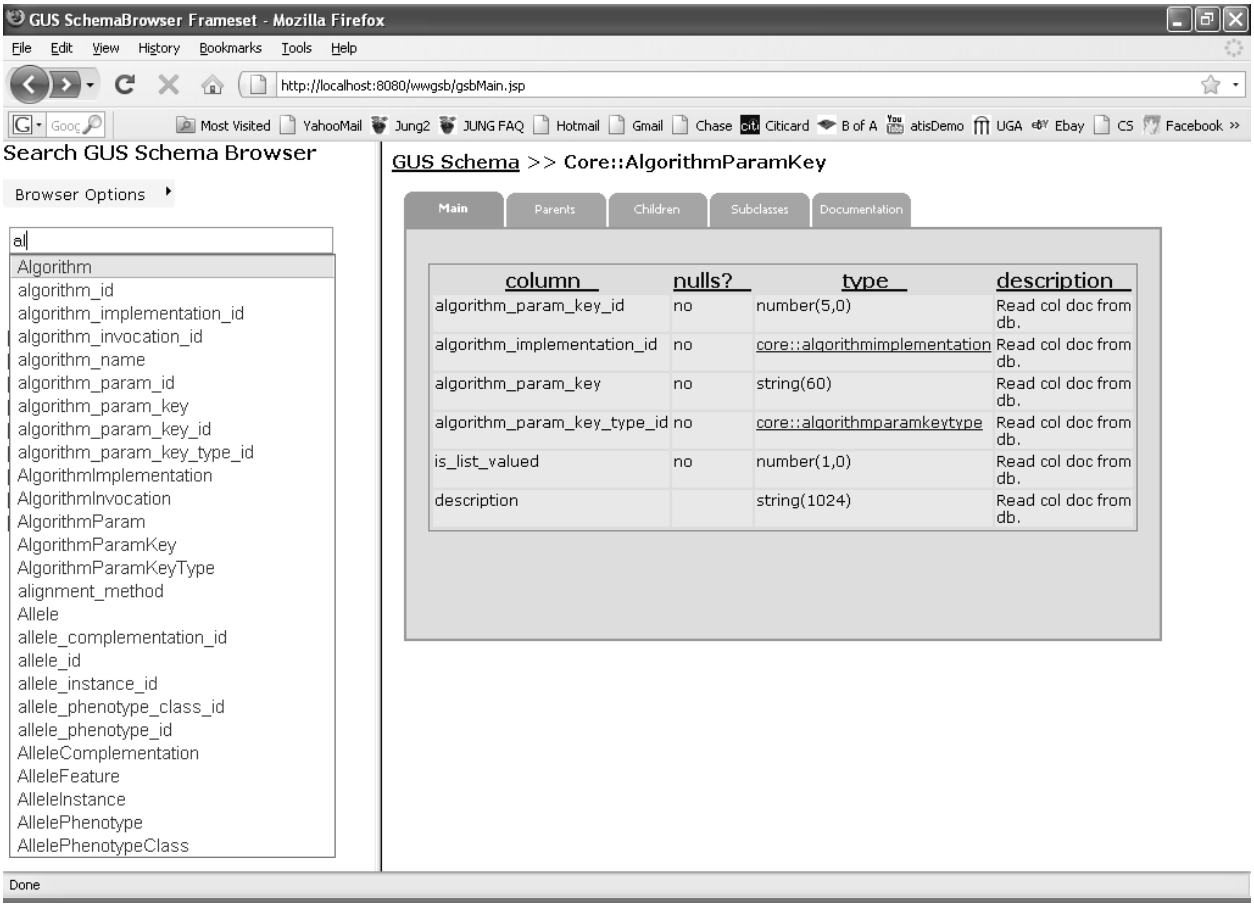

Figure 3.9: Predictive Text Search Functionality

CHAPTER 4

GUS SB - RELATIONSHIP VIEWER

The Online GUS Schema Browser does not provide an effective way for users to explore relationships among the tables of the GUS Schema due to its layout design and lack of navigation controls. Most of all, it does not display a graphical representation of the GUS schema. Therefore, we introduce the Relationship Viewer module for the GUS Schema Browser. The Relationship Viewer attempts to create a model of the GUS Schema in the form of a graph in which the nodes are the tables and the edges represent the foreign key relationships.

4.1 Relationship Viewer Basic Layout

The controls for the Relationship Viewer can be found on the top pane of the window. The graph representing the tables and their relationships can be found on the bottom pane as seen in Figure 4.1. Users can choose to view all tables of GUS, all tables of a single schema, or just one table at a time.

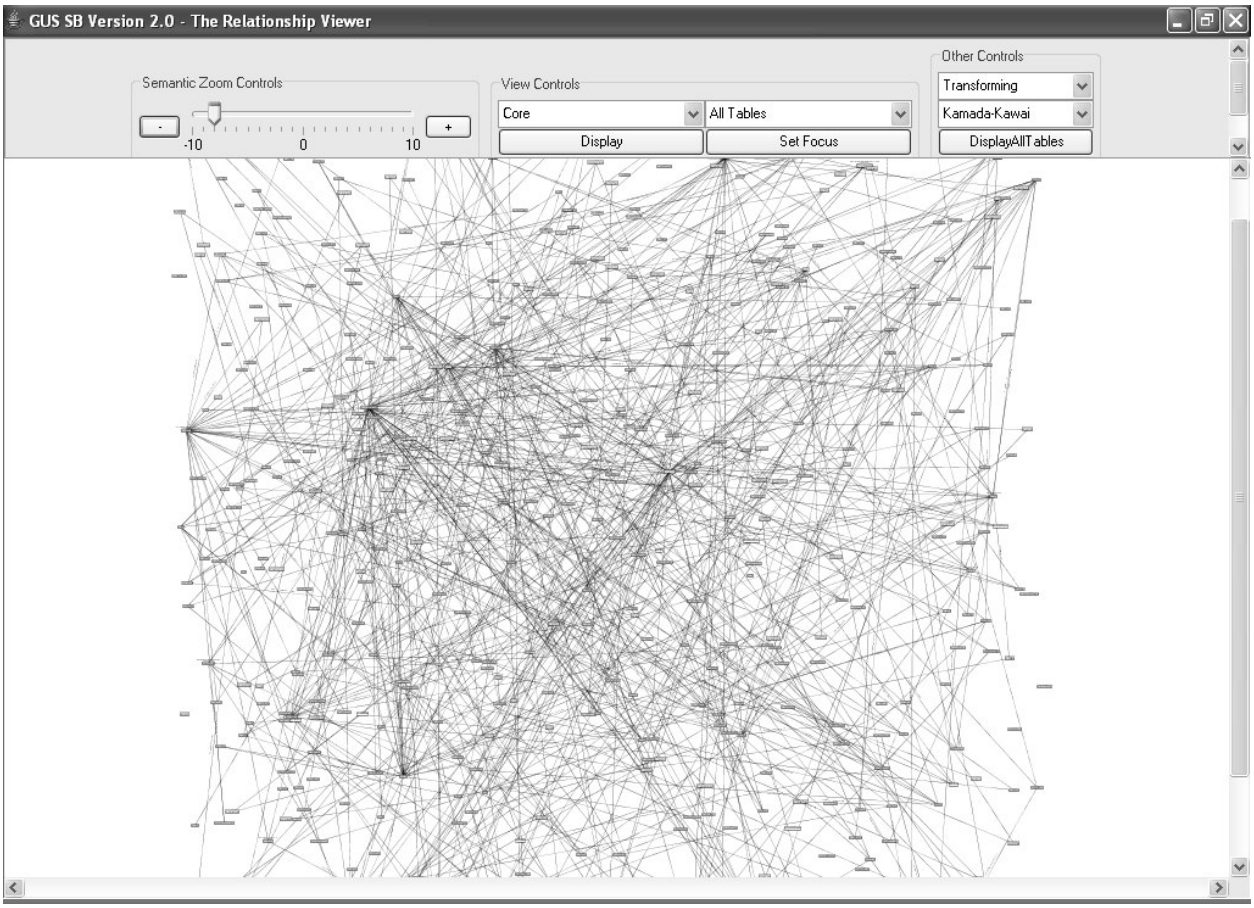

Figure 4.1: The Relationship Viewer Showing All Tables in GUS

4.2 View Controls

Due to the large number of tables in the GUS Schema, the size of a graph generated representing all GUS tables can become very large to the point that not much meaningful information can be obtained from the graph itself. In addition, exploring the entire schema puts a strain on system resources as indicated by noticeable slowdown of browsing capabilities. We attempt to address the issues encountered with very large graphs by allowing the user to locate a particular table in the graph by hitting the 'Set Focus' button, by showing table information in tool tips for mouse-over functionality, and allowing the user to indicate incoming and outgoing edges from a table by changing the colors of the edge. Furthermore, we attempt to decrease the

size of graphs that are generated by allowing the user to display graphs showing only tables and relationships within one particular schema or just a single table. Figure 4.2 shows all the tables in the DoTS schema. Showing a graphical representation of one schema presents a smaller graph and makes it easier for users to browse the graph, isolate single tables, and explore relationships. Table nodes with lots of edges are more visible and may indicate a higher level of importance among other tables.

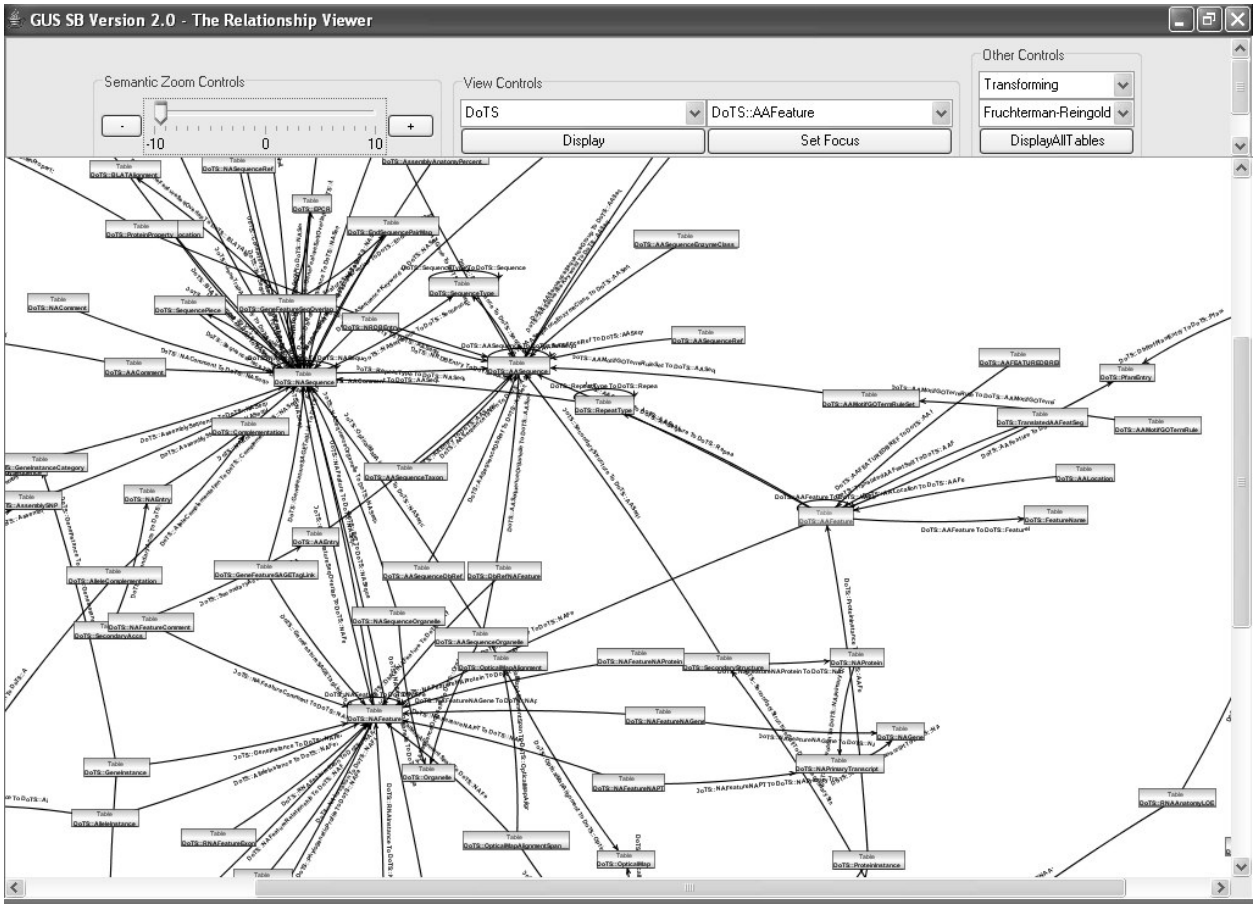

Figure 4.2: The Relationship Viewer Showing All Tables in DoTS

4.3 Semantic Zoom Feature

In most modeling tools, the zooming functionality does not change the underlying data that is being displayed. Due to the size of the graphs that are generated when all tables in the GUS Schema are being displayed, a semantic zoom feature is useful to have. We have attempted to create the effect of semantic zooming in the Relationship Viewer. Currently, there are two levels of table detail that provide the semantic zoom feature. Depending on the zoom level, either all table information on the graph is displayed or only the table name is displayed as seen in Figure 4.3.

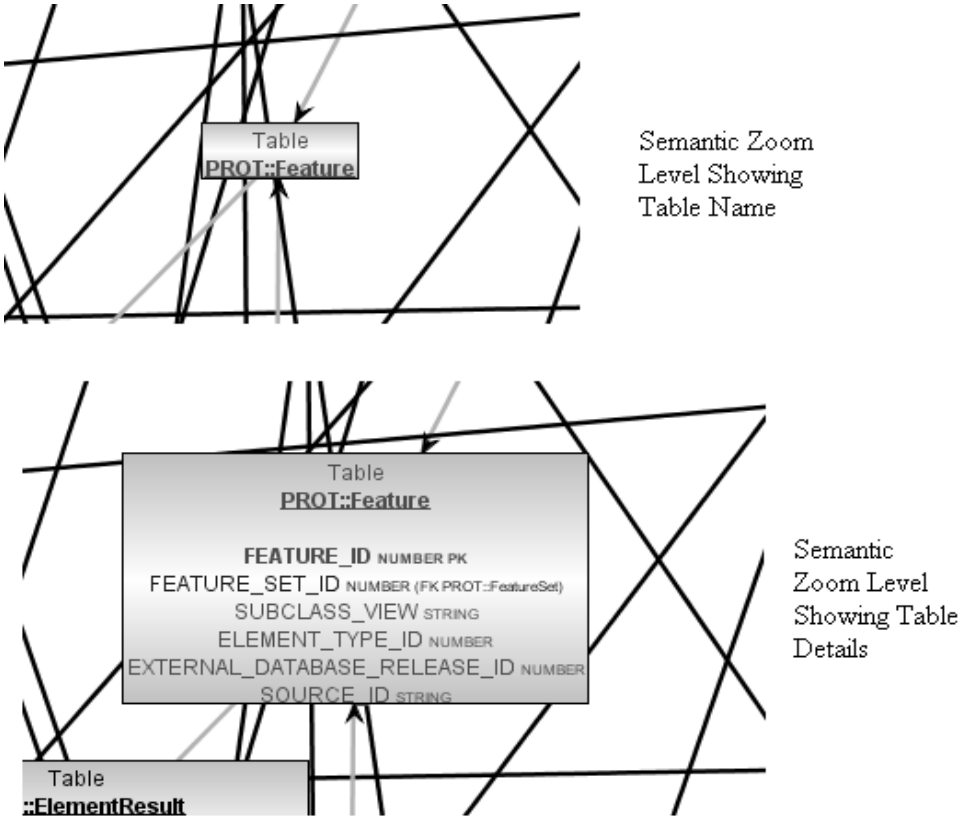

Figure 4.3: Illustration of the Data Displayed Depending on the Semantic Zoom Level

4.4 Single Table View and Relationship Exploration

The Relationship Viewer allows users to isolate a single table in a graph and to add nodes by expanding the relationships of that table. As the graph continues to grow, users can expand the relationships of newly added tables until all foreign key relationships have been exhausted among all the tables in the graph. Expansion of the foreign key relationships among the tables is limited to the child to parent table relationships only. This means that the graph gets larger by evaluating only outgoing edges that represent foreign key relationships to other tables. For now, we also limit the degree level of expansion for a table to ten degrees, as graphs could get extremely large. Figure 4.4 shows the setup of expanding the foreign key relationships for a table by one degree, and Figure 4.5 shows the resulting graph.

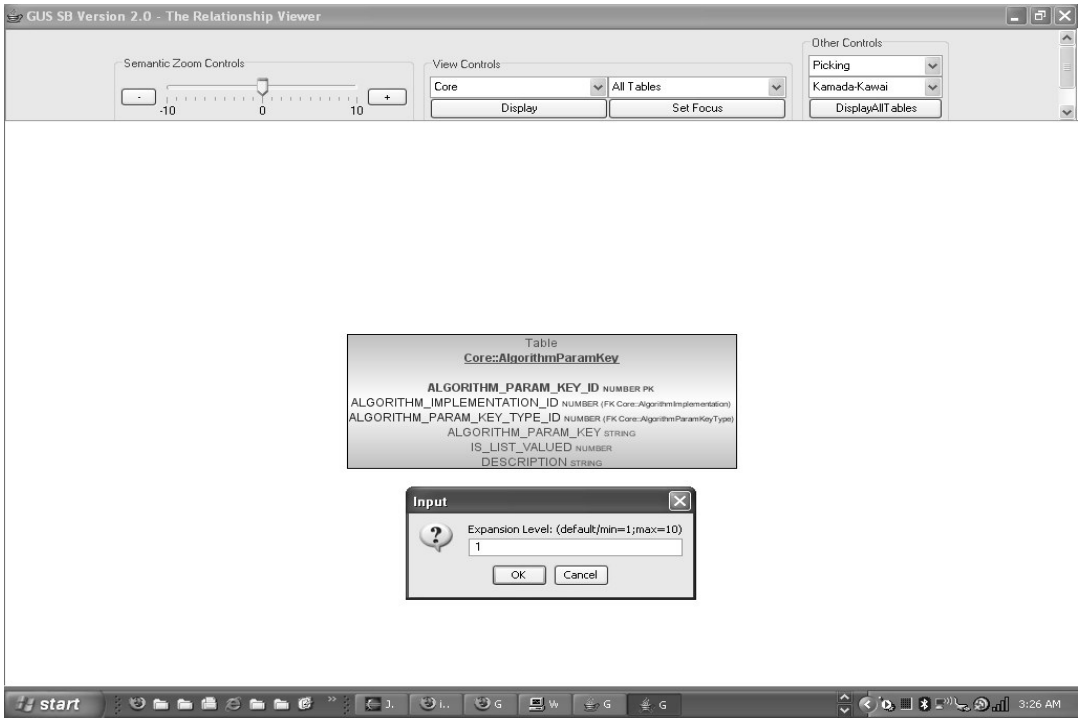

Figure 4.4: Expanding Table Relationships by One Level

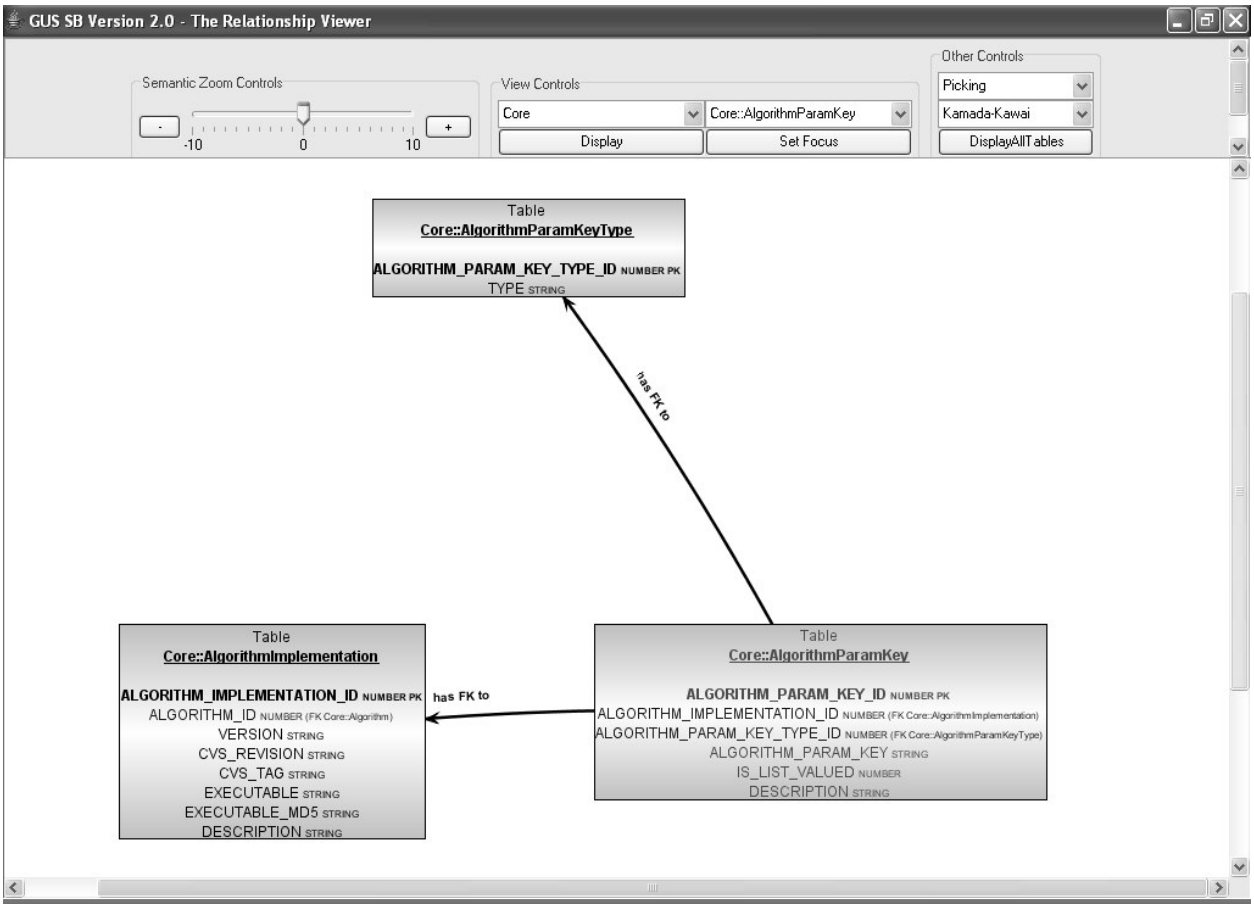

Figure 4.5: Resulting Graph After Expanding Relationships by One Level

4.5 Graph Layouts

The Relationship Viewer has been implemented using JUNG – the Java Universal Network/Graph Framework [41]. The JUNG graphing library comes with several built-in graph layout implementations. Applying layouts to a graph containing all GUS tables takes some time to render, so we use JUNG's persistent graph layout capabilities to generate and save static layout information when viewing all GUS tables.

For this project, we utilize the following JUNG graph layouts:

- Kamada-Kawai [42] – Lays out vertices according to a virtual dynamic system where the total spring energy is minimal.
- Fruchterman-Reingold [43] – A spring-embedder model that attempts to produce aesthetically-pleasing graphs by drawing vertices connected by an edge near each other, while making sure vertices are not drawn too close to each other.
- Spring [36] – A force-directed method that models nodes and edges as physical bodies tied with springs
- ISOM [44] – Implementation of Meyer's Self-Organizing Graphs based on a competitive learning algorithm
- Circle – Lays out all the vertices along the circumference of a circle.

We show two figures to illustrate the layout functionality included with JUNG that has been incorporated in GUS SB. Figure 4.6 shows all tables in GUS using Kamada-Kawai layout, and Figure 4.7 shows all tables in GUS using Circle layout, which resembles images created by Schemaball.

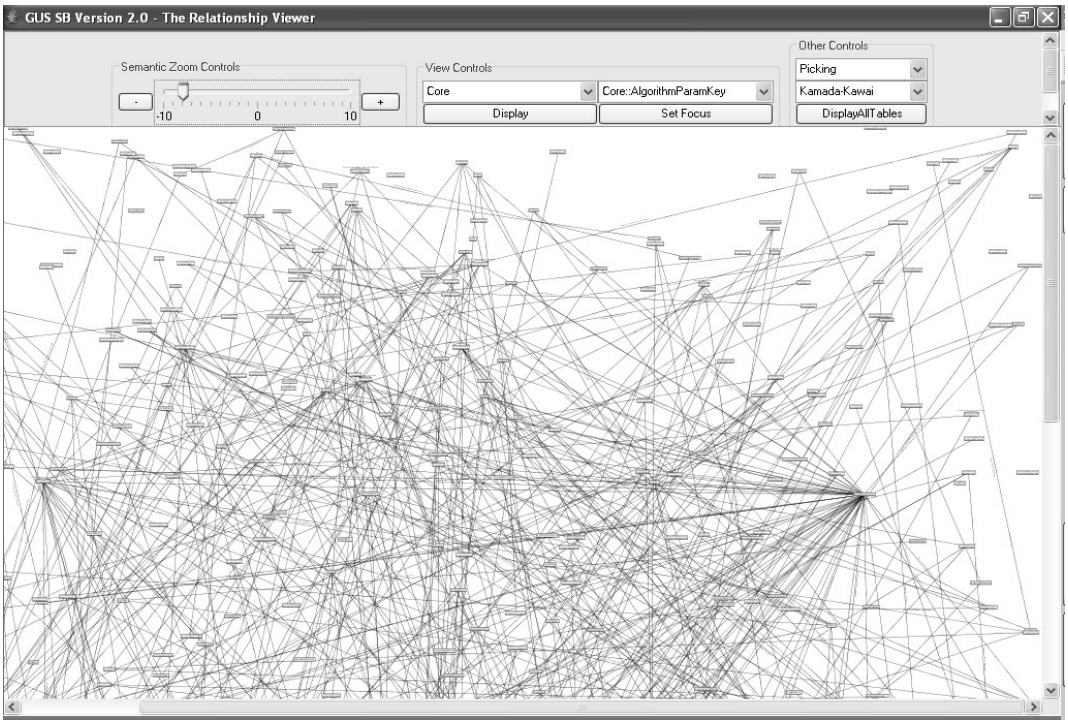

Figure 4.6: All GUS Tables Using Kamada-Kawai Layout

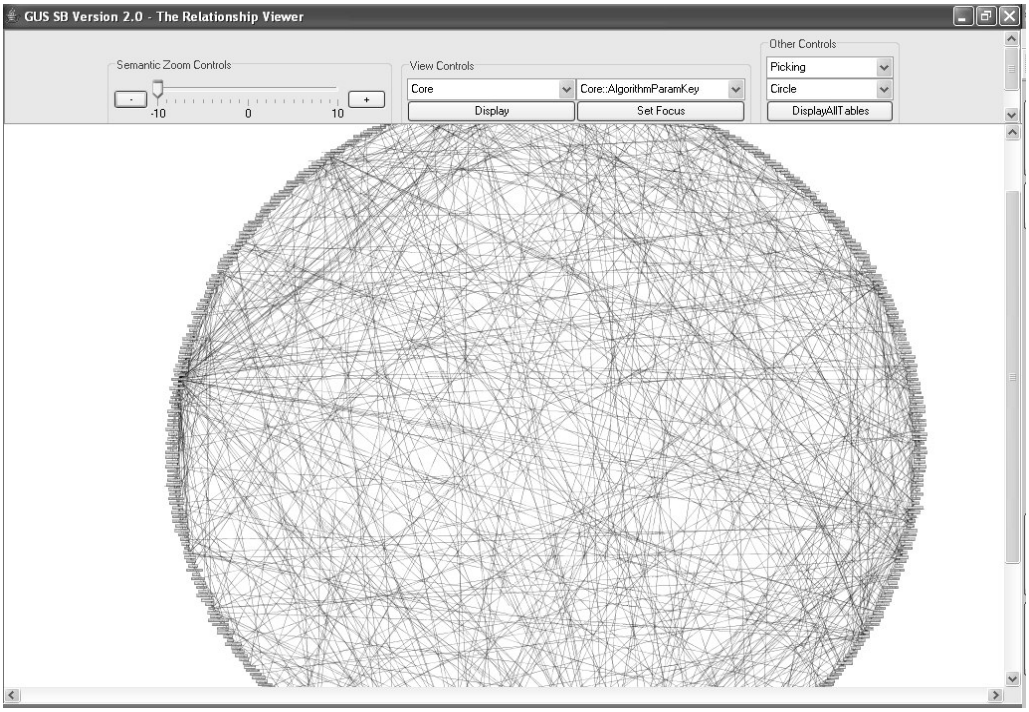

Figure 4.7: All GUS Tables Using Circle Layout

CHAPTER 5

IMPLEMENTATION DETAILS

5.1 Online Schema Browser Enhancements

The architecture of the Online Schema Browser remains the same with our new enhancements. We accomplish the layout changes by using HTML frames. Spring is still used as the Java Model-View-Controller (MVC) framework, but as previously mentioned, new technologies have been added to implement new functionality. We will now go into more details about those technologies.

We have integrated WebWork 2.2 [45], a Java web-application development framework, on top of the Spring MVC. Together with the DOJO toolkit [40], the tabbed browsing for the table information has been implemented. The predictive text (type-ahead) search functionality is accomplished with AJAX calls. To create the navigation trees and menu lists, we use JSCookTree [46] and JSCookMenu [47] respectively, which are written in JavaScript and were created by Heng Yuan.

5.2 GUS SB - Schema Browser

The GUS Schema Browser uses Java 1.5 and the Swing widget toolkit to generate the graphical user interface. GUS SB obtains schema information from GUS XML specification files. It parses data from the files and then stores the information in Java objects. See Appendix A and Appendix B to see samples of the XML files containing GUS schema details. Information is retrieved from the Java objects and is processed for display in the user interface for both the

schema browser and relationship viewer modules. Figure 5.1 shows the architecture for GUS SB.

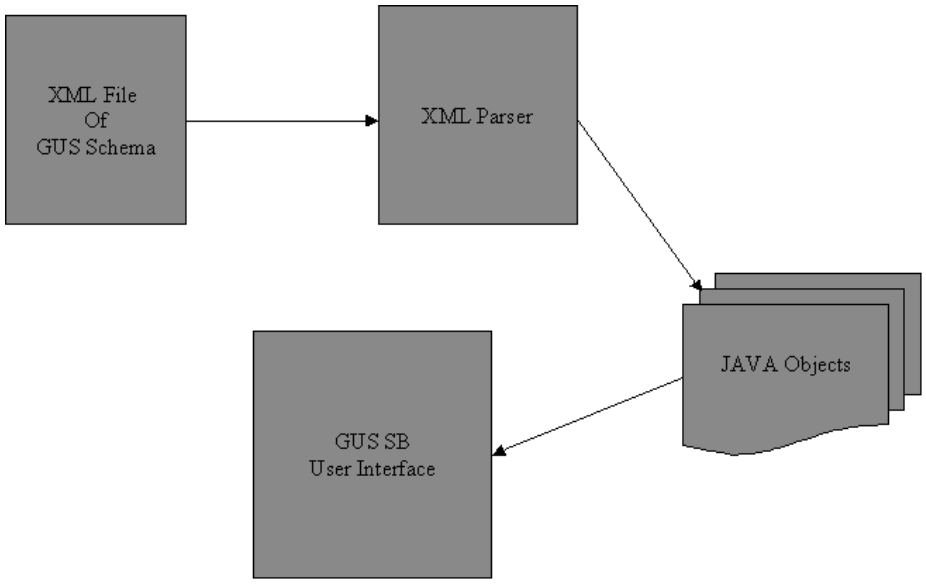

Figure 5.1: GUS SB Architecture

5.3 GUS SB – Relationship Viewer

As already mentioned, the Relationship Viewer has been implemented using JUNG – the Java Universal Network/Graph Framework. JUNG is an open-source library and has been used as the framework for many graph/network analysis and visualization projects, including RDF Gravity, The Graph Exploration System (GUESS), InfoVis Cyberinfrastructure, and GraphExplore [41]. Many more applications and a list of papers describing the use of JUNG in research topics can be found at its website. The Relationship Viewer also uses some aspects of the Piccolo toolkit [48].

We rely extensively on the JUNG framework and its ease of customization to accomplish many of the Relationship Viewer's functionality. In order to display the graph nodes in

rectangular shapes with GUS table information, we have customized how JUNG renders the vertex of the graphs. As previously described, for generating aesthetically pleasing graphs, we utilize the graph layouts included with the framework. To prevent application performance degradation when displaying all tables of GUS and applying layout algorithms, we use saved layout files.

CHAPTER 6

CASE STUDIES AND EVALUATION

6.1 Case Study 1 – Basic Relationship Exploration

We can illustrate the effectiveness of the GUS Schema Browser versus the Online Schema Browser with a simple example. We begin with the Online GUS Schema Browser and choose the Core::AlgorithmParamKey table. We attempt to exhaustively search related tables by evaluating its foreign keys, as represented by HTML links. We see that the table has two foreign keys, Core::AlgorithmImplementation and Core::AlgorithmParamKeyType. We follow a depth-first method for searching and choose the first foreign key by clicking on its link. We see that Core::AlgorithmImplementation has one foreign key, Core::Algorithm. We click on a link to display that table and find it has no foreign keys. We are done with this branch, so we go back to Core::AlgorithmParamKey and evaluate its second foreign key, Core::AlgorithmParamKeyType, by clicking on its link. We find that it has no foreign keys, and we are done with our exhaustive search. It appears we have identified four tables but we might be left with a vague idea of their relationships. Figure 6.1 illustrates the likely result of our exhaustive search of foreign key relationships from child to parent when using the Online GUS Schema Browser. It is probable that we lose track of the relationships that we uncover using the basic link navigation within the Online GUS Schema Browser.

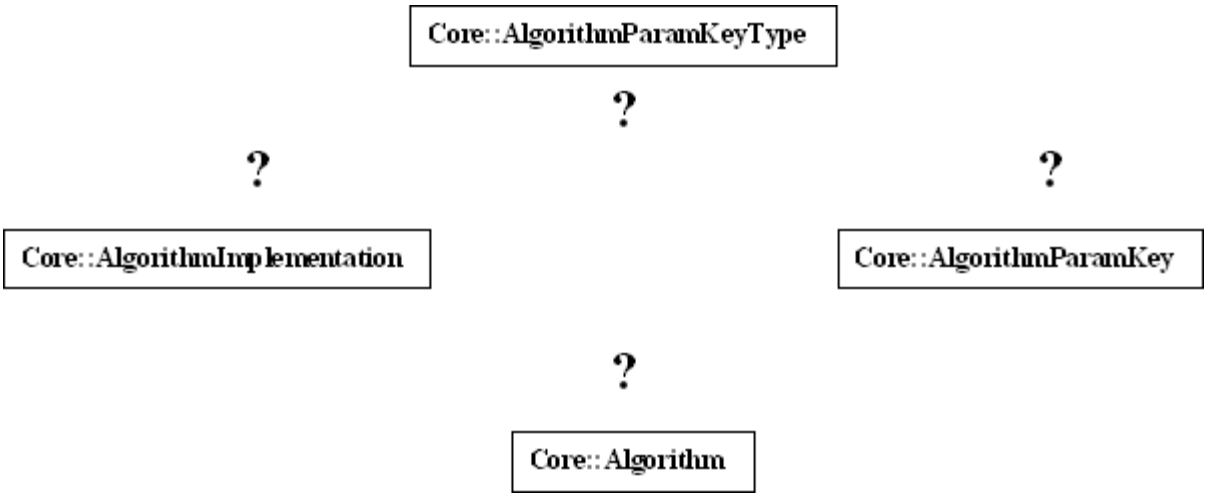

Figure 6.1: Possible Results of Browsing Relationships in Online GUS Schema Browser

Now using the GUS Schema Browser and the relationship exploration functionality of its Relationship Viewer, we can isolate Core::AlgorithmParamKey and begin expanding its relationships and the relationships of subsequent related tables by applying a breadth-first search method. Eventually, we will generate a graph as seen in Figure 6.2 where the relationships among the tables are clearly shown.

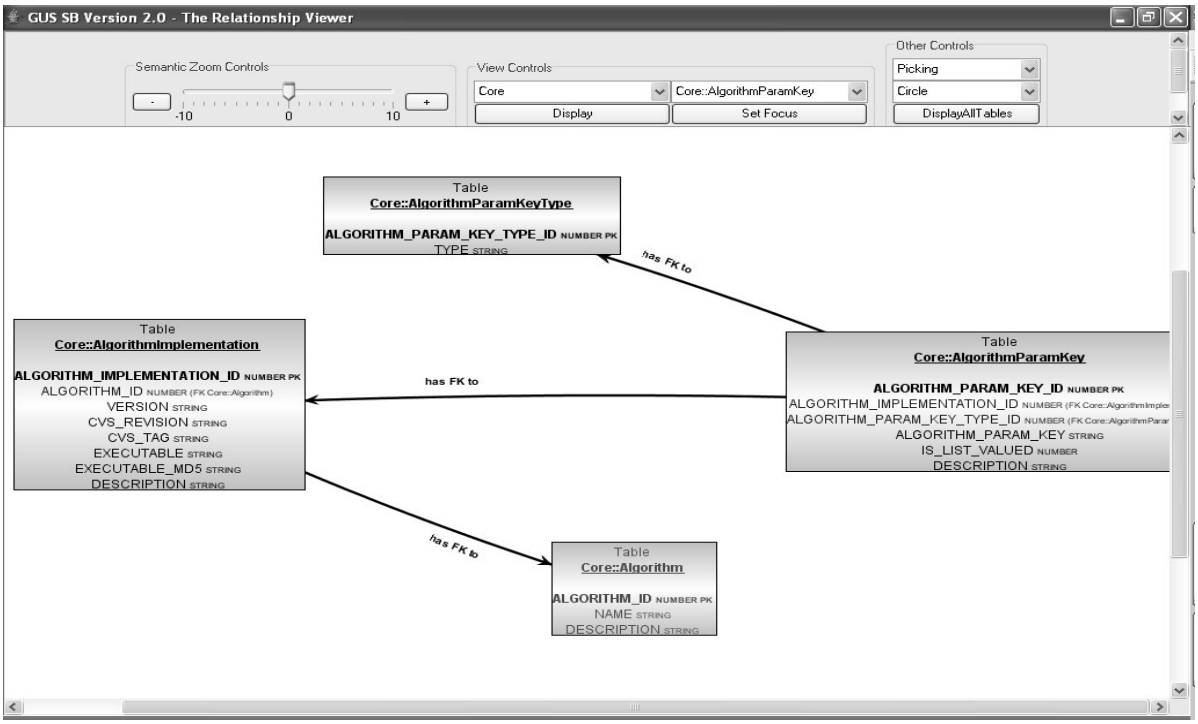

Figure 6.2: Results of Browsing Relationships With GUS Schema Browser

6.2 Case Study 2 – Advanced Relationship Exploration

We can illustrate the effectiveness of the GUS Schema Browser against the diagrams generated by SQL Developer version 2.3.0. We will use DoTS::AAFeature as our table for this example. Our goal is to expand the relationships for DoTS::AAFeature by two degrees, so our objective will be to view all the parent tables for DoTS::AAFeature and then view those tables' parent tables as well.

We begin with SQL Developer and use it to create a diagram. SQL Developer creates a diagram based on the schema and corresponding tables that we choose. We cannot just choose one table and begin exploring relationships interactively. Since we have no idea what foreign keys DoTS::AAFeature contains, we create a graph to include all schemas and all tables in GUS. We isolate DoTS::AAFeature and begin tracing the outgoing edges to other tables. This proves

rather difficult since all tables of GUS are shown and there are many edges. Also, SQL Developer does not implement a semantic zoom feature so table information causes clutter in the graph. See Figure 6.3. Perhaps an easier but still ineffective way to generate the diagram that we desire is to query the database system tables to first identify the parent tables of DoTS::AAFeature and then create another query to identify the parent tables of those tables. In doing so, we manually generate a list of tables that we want in our graph so that we can use SQL Developer to generate the diagram containing only those tables and their relationships. This process takes some time.

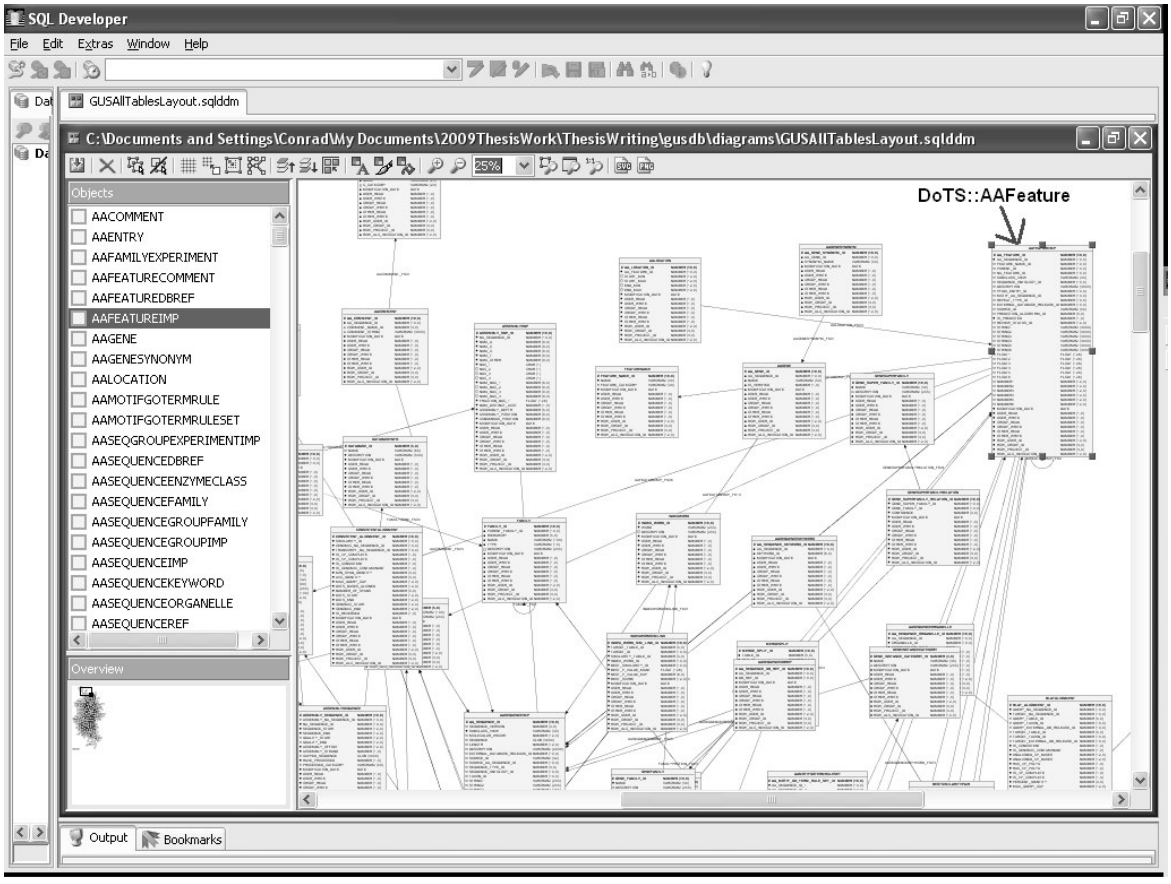

Figure 6.3: SQL Developer Diagram Showing DoTS::AAFeature Among All GUS Tables

Using GUS SB's Relationship Viewer, we can generate an aesthetically pleasing graph showing the information that we want with little effort. First, we need to isolate the DoTS::AAFeature table. We can do this by searching for it in the navigation tree of GUS SB's schema browser module and then clicking on the button to view the table in the Relationship Viewer. In the Relationship Viewer window, we right-click on the table and choose to expand the relationship by two levels. After adjusting the semantic zoom level to view only the table names, we get a neat graph similar to Figure 6.4, showing DoTS::AAFeature, its parent tables, and the parents of those parent tables.

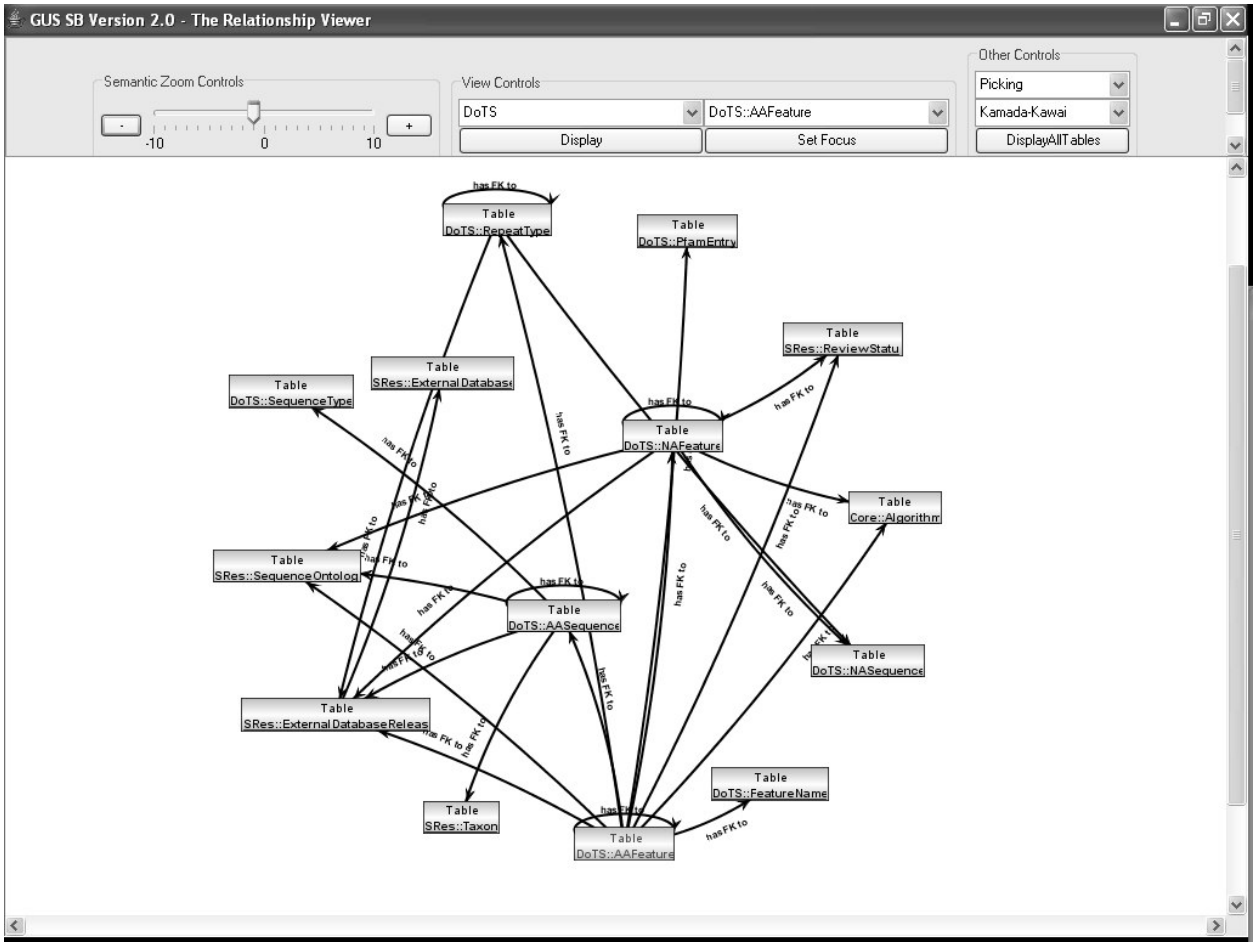

Figure 6.4: Relationship Viewer Showing DoTS::AAFeature Expanded Two Levels

6.3 User Evaluation

We conducted informal interviews with two people who are currently using GUS. One person has been using GUS for more than five years and is very familiar with the GUS schema. Because of his experience, GUS SB is not an extremely helpful tool for his work. The experienced developer is able to browse GUS by creating and executing SQL queries with ease. The second person is relatively new to GUS and is more concerned with data in the tables and the relationships among the data rather than the tables within the GUS schema. Thus, GUS SB would not be extremely useful for this developer's work since he is more concerned with instance data retrieval and analysis. However, the inexperienced GUS developer did say that the GUS Schema Browser would be helpful for learning the schema and would probably be more beneficial to application programmers responsible for loading data into GUS. GUS SB would be able to help those individuals with identifying in which tables and attributes certain data should be stored.

CHAPTER 7

CONCLUSION AND FUTURE WORK

7.1 Conclusion

The new stand-alone GUS Schema Browser, along with its relationship viewer, provides current and potential users of the GUS schema with a more effective way to view tables and their relationships in contrast to the Online Schema Browser that is available today. The Online Schema Browser helps the user understand tables and their attributes but does not facilitate exploring relationships among them. The stand-alone GUS Schema Browser provides the missing functionality to browse GUS tables and their relationships. It also allows for some user interaction in its graphs, which is an advantage over the static diagrams that are generated by some database development tools. As demonstrated in our case studies, GUS SB introduces new features, such as semantic zooming and relationship exploration to allow users to effectively browse the schema. GUS SB can prove to be a valuable tool for new users of GUS or potential users who want explore the schema.

7.2 Future Work

The new version of the stand-alone GUS Schema Browser with its Relationship Viewer should be tested more thoroughly and then released to the public. This will help potential users evaluate the use of GUS versus other genomics schemas since they can browse the schema effectively without having to install GUS into a database or obtain new software. GUS

SB should be released as open source software. There is the possibility that GUS SB can be converted into a generic schema browser for use with other schemas. One feature that should be implemented before GUS SB is released is to support the notion of superclasses and subclasses that are characteristics of the GUS schema. Tables that are subclasses inherit attributes of superclasses and are not depicted accurately by GUS SB at this time. Furthermore, XML schema specification files should be updated and database connection to schema documentation should be considered upon deployment of enhancements to the Online GUS Schema Browser.

The potential exists to develop more functionality within the Relationship Viewer module of GUS SB. Currently only the foreign key relationships between child to parent are utilized. There may be value in showing relationships of parent to child and in supporting the super category/category specifications in GUS. Currently, the GUS Schema Browser and the Relationship Viewer display information in separate windows. Perhaps integrating the two modules into a single window application may be a better design for improving interaction and control.

Future work includes deploying the changes that have been implemented as part of this project to the current Online GUS Schema Browser. The possibility of a web-based GUS Relationship Viewer implementation should be explored. For now, perhaps the stand alone GUS SB can be deployed online as a Java applet.

REFERENCES

[1] http://www.gusdb.org

[2] http://www.ensembl.org/index.html

[3] http://gmod.org/wiki/Main_Page

[4] http://www.springsource.org/

[5] https://www.hibernate.org/

[6] http://mkweb.bcgsc.ca/schemaball/

[7] Cemosek, Gary. "The Value of Modeling." A technical discussion of software modeling : June 2004 < ftp://ftp.software.ibm.com/software/rational/web/whitepapers/ValueOfModeling.pdf>.

[8] Scalzo, Bert. "Why Modeling is Still Relevant." : Information Management Direct: May 2009 < http://www.information-management.com/infodirect/2009_123/data_modeling_databases_management-10015423-1.html>.

[9] Chen, Peter Pin-Shan. The Entity-Relationship Model – Toward a Unified View of Data. ACM Transaction on Database Systems, 1(1):9-36, March 1976.

[10] Rogers, T.R., and R.G.G. Cattell. Entity-Relationship Database User Intefaces: Information Management Group, Sun Microsystems, Incorporated.

[11] http://staruml.sourceforge.net/

[12] http://argouml.tigris.org/

[13] http://projects.gnome.org/dia/

[14] http://office.microsoft.com/visio

[15] http://www-01.ibm.com/software/awdtools/developer/rose/modeler/

[16] http://www-01.ibm.com/software/awdtools/modeler/swmodeler/index.html

[17] http://www.uml.org/

[18] http://squirrel-sql.sourceforge.net/

[19] http://pklite.sourceforge.net/

[20] http://www.phpmyadmin.net/home_page/

[21] http://www.allroundautomations.com/plsqldev.html

[22] http://www.oracle.com/technology/products/database/sql_developer/index.html

[23] http://www.toadsoft.com/

[24] Motro, Amihai. BAROQUE: A Browser for Relational Databases. ACM Transactions on Office Information Systems, 4(2), April 1986

[25] Inder, Robert, and Jussi Stader. Bags and Viewers: A Metaphor For Intelligent Database Access. Advanced Visual Interfaces (AVI'94), Bari, Italy, January 1994.

[26] D'Atri, Alessandro, Amihai Motro, and Laura Tarantino. ViewFinder : An Object Browser. Technical Report ISSE-TR-95-115. February 1995.

[27] http://www.quest.com/toad-data-modeler/

[28] http://www.minq.se/products/dbvis/

[29] Product Family Brief: CA Erwin Modeling Family – At the Center of Your Data Management Initiatives. <http://www.ca.com/files/ProductBriefs/ca-erwin-model-family-prod-family-brief_145815.pdf>

[30] http://www.datanamic.com/dezign/index.html

[31] http://sqldeveloper.solyp.com/

[32] http://schemaspy.sourceforge.net/

[33] Gruber, T. A Translation Approach to Portable Ontologies. Knowledge Acquisition, 1993. 5(2).

[34] http://protege.stanford.edu

[35] Noy, Natalya F., Michael Sintek, Stefan Decker, Monica Crubézy, Ray W.Fergerson, and Mark A. Musen. Creating Semantic Web Contents with Protégé-2000. IEEE Intelligent Systems, 60-71. March/April 2001.

[36] Herman, Ivan, Guy Melancon, and M. Scott Marshall. Graph Visualization and Navigation in Information Visualization: a Survey. IEEE Transaction on Visualization and Computer Graphics, Vol. 6. 2000.

[37] Cockburn, Andy, Amy Karlson, and Benjamin B. Bederson. A Review of Overview+Detail, Zooming, and Focus+Context Interfaces. ACM Comput. Surv. 41, 1, Article2. December 2008. 31 pages.

[38] Catarci, Tiziana, Maria F. Costabile, Stefano Levialdi, Carlo Batini. Visual Query Systems for Databases: A Survey. 1995

[39] Krzyvinski, Martin. Schemaball: A New Spin on Database Visualization. SysAdmin Journal. August 2004. Vol. 13 Issue 08.

[40] http://www.dojotoolkit.org/

[41] http://jung.sourceforge.net/

[42] Kamada, Tomihisa, and Satoru Kawai. An Algorithm for Drawing General Undirected Graphs. Information Processing Letters. 31. 7-15. 1989

[43] Fruchterman, Thomas M.J., and Edward M. Reingold. Graph Drawing by Force-directed Placement. Software-Practice and Experience, Vol. 21(1 1), 1129-1164. November 1991

[44] Meyer, Bernd. Self-Organizing Graphs A Neural Network Perspective of Graph Layout. Graph Drawing. August 1998.

[45] http://www.opensymphony.com/webwork/

[46] http://jscook.yuanheng.org/JSCookTree/

[47] http://jscook.yuanheng.org/JSCookMenu/

[48] http://www.cs.umd.edu/hcil/jazz/

APPENDIX A

SNIPPET OF XML FILE CONTAINING GUS SCHEMA INFORMATION


```
<?xml version="1.0"?>
<database name="null"><schemas>
<schema name="Core">
<tables>
<table  id="Core/Algorithm" name="Algorithm" housekeeping="true" versioned="true"
tablespace="USERS" categoryRef="Algorithm" updatable="true">
<columns>
<column id="Core/Algorithm/ALGORITHM_ID" name="ALGORITHM_ID" nullable="false"
length="5" precision="0" type="NUMBER"/>
<column id="Core/Algorithm/NAME" name="NAME" nullable="false" length="100"
precision="0" type="STRING"/>
<column id="Core/Algorithm/DESCRIPTION" name="DESCRIPTION" nullable="true"
length="255" precision="0" type="STRING"/>
</columns>
<indexes>
<index name="PK_ALGORITHM" tablespace="USERS" type="NORMAL">
<columns>
<column idref="Core/Algorithm/ALGORITHM_ID"/>
</columns>
</index>
</indexes>
<constraints>
<constraint name="PK_ALGORITHM" type="PRIMARY_KEY">
<constrainedColumns>
<column idref="Core/Algorithm/ALGORITHM_ID"/>
</constrainedColumns>
</constraint>
</constraints>
</table>
<table id="Core/AlgorithmImplementation" name="AlgorithmImplementation"
housekeeping="true" versioned="true" tablespace="USERS" categoryRef="Algorithm"
updatable="true">
<columns>
<column id="Core/AlgorithmImplementation/ALGORITHM_IMPLEMENTATION_ID"
name="ALGORITHM_IMPLEMENTATION_ID" nullable="false" length="5" precision="0"
type="NUMBER"/>
<column id="Core/AlgorithmImplementation/ALGORITHM_ID" name="ALGORITHM_ID"
nullable="false" length="5" precision="0" type="NUMBER"/>
```

<column id="Core/AlgorithmImplementation/VERSION" name="VERSION" nullable="true" length="10" precision="0" type="STRING"/>
<column id="Core/AlgorithmImplementation/CVS_REVISION" name="CVS_REVISION" nullable="true" length="20" precision="0" type="STRING"/>
<column id="Core/AlgorithmImplementation/CVS_TAG" name="CVS_TAG" nullable="true" length="100" precision="0" type="STRING"/>
<column id="Core/AlgorithmImplementation/EXECUTABLE" name="EXECUTABLE" nullable="true" length="255" precision="0" type="STRING"/>
<column id="Core/AlgorithmImplementation/EXECUTABLE_MD5" name="EXECUTABLE_MD5" nullable="true" length="32" precision="0" type="STRING"/>
<column id="Core/AlgorithmImplementation/DESCRIPTION" name="DESCRIPTION" nullable="true" length="500" precision="0" type="STRING"/>
</columns>
<indexes>
<index name="ALGORITHMIMPLEMENTATION_IND01" tablespace="USERS" type="NORMAL">
<columns>
<column idref="Core/AlgorithmImplementation/ALGORITHM_ID"/>
</columns>
</index>
<index name="PK_ALGORITHMIMPLEMENTATION" tablespace="USERS" type="NORMAL">
<columns>
<column idref="Core/AlgorithmImplementation/ALGORITHM_IMPLEMENTATION_ID"/>
</columns>
</indexes>
<constraints>
<constraint name="ALGORITHMIMPLEMENTATION_FK04" type="FOREIGN_KEY">
<constrainedColumns>
<column idref="Core/AlgorithmImplementation/ALGORITHM_ID"/>
</constrainedColumns>
<referencedTable idref="Core/Algorithm"/>
<referencedColumns>
<column idref="Core/Algorithm/ALGORITHM_ID"/>
</referencedColumns>
</constraint>
<constraint name="PK_ALGORITHMIMPLEMENTATION" type="PRIMARY_KEY">
<constrainedColumns>
<column idref="Core/AlgorithmImplementation/ALGORITHM_IMPLEMENTATION_ID"/>
</constrainedColumns>
</constraint>
</constraints>
</table>

APPENDIX B

XML FILE CONTAINING GUS SCHEMA CATEGORY INFORMATION

```
<?xml version="1.0"?>
<organization>
    <supercategory name="Sequence and Features">
      <category name="NA Sequence"/>
      <category name="AA Sequence"/>
      <category name="NA Sequence Features"/>
      <category name="AA Sequence Features"/>
      <category name="Feature Relations"/>
      <category name="GenBank Sequence Record"/>
      <category name="Similarity"/>
      <category name="Assembly"/>
      <category name="Motifs"/>
    </supercategory>
    <supercategory name="Function">
      <category name="Central Dogma"/>
      <category name="Paralog and Family"/>
      <category name="Sequence Ortholog, Paralog, Family"/>
      <category name="AA Ortholog"/>
      <category name="Mapping"/>
      <category name="Clones"/>
      <category name="Interaction"/>
      <category name="GO Association"/>
      <category name="Variation"/>
      <category name="Raw Mass Spectrometry Results"/>
      <category name="Protein Identification (From Mass Spec Analysis)"/>
    </supercategory>
    <supercategory name="Transcription Regulation">
      <category name="Regulation Framework"/>
      <category name="Regulatory Moieties"/>
      <category name="Regulatory Activities"/>
      <category name="Comments"/>
      <category name="Models"/>
      <category name="Multinomial Models"/>
      <category name="Families of Similar Models"/>
      <category name="Occam's Razor"/>
      <category name="High Volume Genome Annotation"/>
      <category name="Training Sets"/>
```

```
</supercategory>
      <supercategory name="Experiment">
         <category name="Protocols"/>
         <category name="Data Transformations and Analyses"/>
         <category name="Learning Models"/>
         <category name="Model-based Regulatory Feature Prediction"/>
         <category name="Bounded Collection Grammars"/>
         <category name="Platform"/>
         <category name="Assay"/>
         <category name="Quantified Data"/>
         <category name="Experimental Design"/>
         <category name="Biomaterials"/>
         <category name="(Mixed) Ontologies"/>
         <category name="Integrity"/>
      </supercategory>
      <supercategory name="Provenance">
         <category name="Algorithm"/>
         <category name="External Database"/>
         <category name="Bibliographic"/>
         <category name="Evidence"/>
      </supercategory>
      <supercategory name="Vocabularies">
         <category name="Anatomy Vocabulary"/>
         <category name="Gene Ontology"/>
         <category name="Enzymes Vocabulary"/>
         <category name="Taxon Vocabulary"/>
         <category name="Developmental Stage Vocabulary"/>
         <category name="Disease Vocabulary"/>
         <category name="Genetic Code Vocabulary"/>
         <category name="Generic Ontology"/>
         <category name="Mutagens Vocabulary"/>
         <category name="Phenotype Vocabulary"/>
         <category name="Sequence Ontology"/>
         <category name="Repeat Types Vocabulary"/>
         <category name="Sequence Types Vocabulary"/>
      </supercategory>
      <supercategory name="Administration">
         <category name="Meta Info"/>
         <category name="Administration"/>
         <category name="Misc Applications"/>
         <category name="Text Search"/>
      </supercategory>
</organization>
```