ABSTRACT

Motivation: To improve the ability of biologists (both researchers and students) to ask biologically interesting questions of the Gene Ontology (GO) database and to explore the ontologies by seeing large portions of the ontology graphs in context, along with details of individual terms in the ontologies.

Results: GoGet and GoView are two new tools built as part of an extensible web application system based on Java 2 Enterprise Edition technology. GoGet has a user interface that enables users to ask biologically interesting questions, such as (1) What are the DNA binding proteins involved in DNA repair, but not in DNA replication? and (2) Of the terms containing the word triphosphatase, which have associated gene products from mouse, but not fruit fly? The results of such queries can be viewed in a collapsed tabular format that eases the burden of getting through large tables of data. GoView enables users to explore the large directed acyclic graph structure of the ontologies in the GO database. The two tools are coordinated, so that results from queries in GoGet can be visualized in GoView in the ontology in which they appear, and explorations started from GoView can request details of gene product associations to appear in a result table in GoGet.

Availability: Free access to the GoGet query tool and free download of the GoView ontology viewer are provided to all users at http://db.math.macalester.edu/goproject. In addition, source code for the GoView tool is also available from this site, along with a user manual for both tools.

Contact: shoop@macalester.edu

INTRODUCTION

The goal of this project was to provide powerful new ways for biologists to explore the Gene Ontology (GO) data that are being developed by the GO Consortium. The GO Consortium was formed to build a common annotation to describe genes and their products in any organism (Ashburner et al., 2000; The Gene Ontology Consortium, 2001; Ashburner and Lewis, 2002). As stated by The Gene Ontology Consortium (2001), one of the consortium’s goals was to create ‘a shared language of biology’ by compiling ‘a comprehensive structured vocabulary of terms describing different elements of molecular biology that are shared among life forms’. The structured vocabularies are organized as ontologies, which are a diagrammatic and notational tool for describing knowledge about a domain (Gruber, 1993, 1995).

The consortium has built and continues to revise three ontologies: molecular function, biological process and cellular component. These are defined by The Gene Ontology Consortium (2001):

‘Briefly, molecular function describes what a gene product does at the biochemical level. Biological process describes a broad biological objective. Cellular component describes the location of a gene product, within cellular structures and within macromolecular complexes’.

Each ontology contains terms, which form a hierarchical directed acyclic graph (DAG). Each term has one or more ‘child’ terms, and a child term can have more than one ‘parent’ term. The relationships between a parent term and a child term are either ‘is-a’ or ‘is-part-of’. An example of the subgraph of the DAG for molecular functions beginning with the amino acid transporter term can be seen in Figure 1, which depicts one of the exploration tools that we have built, called GoView (described in detail in the subsequent sections). The boxes represent the terms and the edges between boxes are the relationships between terms.

In addition to developing the ontologies, the consortium members are assigning gene products from 26 model organism and sequence databases to one or more of the terms in
each ontology. The assignment is based on a particular kind of evidence, including inference by direct assay, by sequence similarity or by traceable author statement, to name a few. The gene products can be associated with more than one term in all three ontologies. These associations and the terms are freely available in relational database format from the consortium at http://www.godatabase.org. As of the July 2003 release, the GO database has 5399 terms with 6557 edges in the molecular function ontology, 7309 terms with 11202 edges in the biological process ontology and 1304 terms with 1644 edges in the cellular component ontology. A total of 1397281 gene products have 8005590 associations in the molecular function ontology, 1251601 gene products have 7236421 associations in the biological process ontology and 1141640 gene products have 6554115 associations in the cellular component ontology.

Several web applications and data access tools for GO data have been developed recently. AmiGO (The Gene Ontology Consortium, 2002, http://www.godatabase.org/cgi-bin/go.cgi) depicts each of the three GO DAGs as a collapsed tree on which users can choose to open and reveal the next level of children. AmiGO also provides a mechanism to perform a text search within either terms or gene products (but not both). GenNav (National Library of Medicine, 2002, http://etbsun2.nlm.nih.gov:8000/perl/gennav.pl) has users start with a text search of terms. Once the user chooses from one of several terms that may get returned, individual paths in the large DAG from the root of the ontology to that term are displayed, including is-a and is-part-of relationships, along with all the other information about that term. The GeneAround GO Viewer (Tanoue et al., 2002) also depicts paths in the GO graphs for selected terms. The GoFish tool (Berriz et al., 2003) enhances the search for gene product associations to terms by ranking gene products based on a score calculation that represents how closely a gene product matches a boolean query of terms.
Confluence of ideas
The GO consortium has stated that ‘the structure of the ontology permits the implementation of robust query capabilities far beyond the development of a simple dictionary of terms and keywords’ (The Gene Ontology Consortium, 2001). We concur with this statement and have developed a novel GO data exploration system that enables researchers and novices to both visualize and interact with large portions of the ontologies and to ask complex, biologically interesting questions about the terms and their associated gene products. To build the system we have drawn on contributions made in information visualization, graph drawing and distributed client/server computing over the Internet.

Information visualization
Card et al. (1999) define information visualization as ‘the use of computer-supported, interactive, visual representations of abstract data to amplify cognition’. Current GO web applications and search tools are interactive and provide visual representations, or views, in the form of trees and paths in the graphs to depict portions of the ontologies. Information visualization techniques also include multiple views on large datasets with additional interactive capabilities such as zooming and panning of large views that are not visible on one monitor. Information visualization research has shown that it is important to provide multiple views of complex data (Baldonado et al., 2000), and that coupling, or linking of those views (when the user changes one view, another view changes accordingly) helps users to understand how information is related when navigating (Buja et al., 1991). When information such as a DAG is large (thousands of nodes), it is sometimes helpful to provide an overall view with details removed, called a context view, and to enable users to view more information about part of the graph in a second detailed view that is coupled to the context view (Beard and Walker, 1990).

Graph drawing
There are a myriad of algorithms to lay out large connected graphs so that they are informative and pleasing to the eye (Battista et al., 1999). The basic premise of all graph drawing algorithms is to take a set of related ‘nodes’ and determine where to place those nodes on a two-dimensional (2D) plane (or possibly higher dimensional surface) and draw ‘edges’ to represent the relationships between the nodes. Most algorithms cannot satisfy all the desired properties of certain graphs such as DAGs, but instead will prioritize among properties like fewest line crossings of the edges, level node placement, uniform edge length, fewest bends in the edges and compactness. Because the GO ontology graphs have term nodes with potentially long names, we chose to use an algorithm that used boxes for the nodes. There are not a large number of levels in any of the three ontology DAGs, but there are often many terms at each level. We therefore chose an algorithm that could draw the graphs by placing the nodes in a line on each level and minimize the number of edge crossings. The algorithm we used was introduced by Gansner et al. (1993) and is freely available as open source from AT&T Labs Research (2003, http://www.research.att.com/sw/tools/graphviz/). A survey of graph drawing algorithms and information visualization techniques for large graphs can be found in Herman et al. (2000).

Client-server computing over the Internet
Much of the previous work on information visualization has been applications that read specially formatted data files or individual tables from a database and keep the data in memory in the application. There is such a large amount of ever-changing biological data that it is impractical to keep it on individual users’ computers. With recent advances in client-server computing on the Internet; however, it is now possible to build applications that keep certain data in memory on a local ‘client’ computer, but also request other data from a remote ‘server’. The client has the code for visualizing the data, and requests certain portions of the data from the server when necessary. If these servers are on the Internet and can handle many simultaneous requests, then many clients can access the data at once.

The emerging standard for Internet-based client-server computing is to use a multi-tiered system that includes a database management system (DBMS) to hold the data, multiple clients to view the data and one or more layers in between (called middleware), in which communication with the database and the client is accomplished in an application server (Bernstein, 1996). This architecture is advantageous for several reasons: (1) all of the network connection code for remote procedure calls (RPC) is hidden inside the application server, (2) the application server can manage multiple database connections efficiently by providing an available ‘pool’ of connections, (3) information about client sessions can be kept with the application server throughout the lifetime of each session and (4) several different kinds of clients can provide separate views on the data but use the same connection and data access services provided by the application server. The two primary competing technologies are those based on Sun Microsystems’ Java 2 Enterprise Edition (J2EE) standard, for which several commercial and free application servers are available, and Microsoft’s .NET developer tools. We have built our system using J2EE because (1) we were building Java applications, (2) we were using Java Server Pages (JSP) in our web applications (Hall, 2002), (3) we have Unix operating systems on our server hardware and (4) we could obtain an open source application server, called JBoss (http://www.jboss.org).
SYSTEM AND METHODS

In this section, we will describe what new kinds of data exploration users can accomplish with our tools. We will present biologically interesting questions and show how the tools can be used to get the answers and explore the GO terms and gene product associations along the way. We will also point out some of the features of the tools. In the next section, we will describe how these tools were built and how the system can expand to include new tools with a fair amount of software reuse. The two main tools that we provide are GoGet, which is a web application tool for formulating biologically interesting queries, and GoView, which is a stand-alone Java application for exploring each of the three ontologies as large DAGs. The tools are linked, so that query results can be mapped to the terms and viewed in GoView, and the details about gene products associated with terms chosen in GoView can be requested and displayed in GoGet.

Querying with GoGet

Many web applications provide a single text box search as the first interface that the user sees. Although easy to use and not intimidating, the user is sometimes left to wonder just how the results were obtained and whether they missed anything that another search would reveal. Worse yet, they might not even be aware that there are data that they have missed. If boolean logic queries are available, they may not use them because they are unfamiliar with the proper format to use (sites vary quite a bit in how boolean queries are formed). When the information is purely text documents, the single-box text string search is what can be offered to users. Because the GO data are so rich with information that is well structured in a relational database, we sought to provide a query interface that would enable biologists to ask interesting questions and to know what they were asking by providing an English sentence version of their complex query as they were creating it.

Figure 2 shows the initial state of GoGet when a user first locates the URL (http://db.math.macalester.edu/GoProject/goGet.jsp).

To ease the user into an understanding of how to build a query, we initially provide one by filling in the upper text box inside the middle left box. This box is called the ‘Set 1’ box, because users can request a set of terms and their gene products to be returned. The box in the lower right shows what the English sentence version of this query is (we call this the ‘My Search’ box).

Each of the check boxes on the interface provides an opportunity for user to filter the amount of information that will be returned. In the very upper left, users can limit which of the three ontologies to search. On the upper right, both data sources that the gene products came from and evidence provided for the association of that gene product with a term can be limited by choosing certain data sources and evidence codes, respectively. We get the information from the database to build this part of the tool dynamically. When data sources change or evidence codes are added or removed with new updates of the GO database, users will see the latest available choices in GoGet. Both the database or evidence code and a description are included so users know what they are choosing. The default is to have all chosen so the results are the most inclusive. Unchecking any of these checkboxes will potentially return fewer results. The checkboxes labeled ‘All’ are toggles—click once when any one was chosen and they will all be cleared, and click once when none was chosen and they will all be chosen.

The buttons along the top of the tool toggle the display of the query formulation boxes (except for search and clear). If a user is satisfied with his/her choices for data sources, for example, he/she can choose to get them out of their way and reduce the cognitive overhead of keeping track of the filtering options chosen. Figure 3 shows the results of the above query, after the user has chosen to search. The terms with triphosphatase somewhere in their name are placed in a table, and the gene products in those terms are initially hidden, with only the count of the total number of gene products being displayed. This is the number of gene products associated with that particular term, not with any of its parents or children in the graph. Users can choose to ‘display’ the details about each of the associated gene products, and the table then gets larger, with a subtable of gene product information shown where the display button was. This enables details to be hidden unless the user really wants to see them, and keeps the table from getting overwhelming to comprehend. If there are many gene products for a term, a user could try to narrow the search by adding text for the gene product’s name, or choosing gene products from certain data sources or those associated to a term with a particular level of evidence. This can be performed by using the buttons at the top to reveal the search box desired.

The ‘Set 1’ and ‘Set 2’ boxes are where complex queries can be formed using text searches on a combination of terms and gene products. As an example of a complex, yet biologically interesting query, the GO Consortium provides this question in The Gene Ontology Consortium (2001):

‘…what are the DNA binding proteins involved in DNA repair, but not in DNA replication’?

Figure 4 shows how this query would be formed in GoGet and the corresponding English version, which helps users to be certain about what they have asked for. Many other combinations of term and gene product searches are possible. This kind of sophisticated search where sets of gene products are compared cannot be performed in any of the current GO web applications.
When a set of terms have been returned, users can perform special additional analyses of the gene products associated with those terms. One analysis is to ask which terms have gene products from one data source AND, OR, BUT NOT another data source (Fig. 5). As the database becomes complete, this will be close to being able to ask how two species differ (BUT NOT), how they are the same (AND), or whether one or the other has associated gene products (OR). The GO database now contains species data for each gene product, which was not the case when we first developed GoGet. We plan to
Fig. 4. The GoGet tool with complex query using both set boxes.

Fig. 5. The Macalester GoGet tool showing terms searched for and formation of a complex query.
change this analysis to use the species of each of the gene products rather than its data source in a future release. Users can choose the set of terms that will be used for an analysis. A second available analysis is to display the gene products that are associated with two or more terms from the set of terms chosen. This would return gene products that are common to several terms. These types of questions cannot be asked using currently available GO web tools.

**Exploring the ontologies with GoView**

GoView is a stand-alone Java application that may be downloaded from [http://db.math.macalester.edu/goproject](http://db.math.macalester.edu/goproject). Rather than using the collapsed tree view that many existing tools use, we surmised that it might be useful to view each ontology as the graph that it actually is and enable users to explore the graph. Since the ontologies are large DAGs that will keep getting larger as the database grows, they cannot be viewed completely on a single computer monitor, nor can human readable graphs be made from any graph drawing algorithm, because of the size that each term box node needs for the term name. Users first must choose a subgraph of a larger ontology—we determined that subgraphs with approximately 1200 edges or less are of reasonable size for exploration.

For example, Figure 1 shows the GoView tool after the user has chosen the ‘physiological processes’ term from within the biological processes ontology, then the ‘metabolism’ child term of physiological processes, and the ‘coenzymes and prosthetic group metabolism’ child term of metabolism. This term then contains a small enough subgraph to explore. The small graph to the left provides the overall context by depicting the whole subgraph. The box on that context view shows what portion of the graph is visible in the area on the right (the detail view). This detail view can be zoomed in or out (using the zoom bar or buttons below the context view) to show smaller or larger sections of the subgraph, and the box in the context view changes size to match the detail view. Users can pan through the whole subgraph either by manipulating the box in the context view, using the mouse in the detail view, or using the scrollbars around the detail view.

Users can explore the ontologies and manipulate each subgraph in several other ways. They can explore more than one ontology subgraph—each one that is chosen from the lists in the top section is placed in a separate tabbed view. These tabs can be closed when the user no longer needs them. They can select single terms by clicking each one or they can select a set of terms by lassoing them with the mouse. (These selections can be used to obtain the information about gene products associated with those terms; see below.)

Portions of graphs can be highlighted or pulled out for ease of exploration. Siblings of a selected term can be indicated. This is useful for seeing and then analyzing all the terms that are children of a term’s parent. Figure 6 shows a term in the coenzymes and prosthetic group metabolism subgraph highlighted (approximately in the middle right), and several of its visible siblings highlighted (shown as light gray boxes and text). In this example, the siblings share the parent term ‘coenzyme biosynthesis’. As much as possible, siblings are at the same level in the graphs, but occasionally this is not possible because of the nature of the ontology graph. When this happens, we provide this feature so that the siblings are easy to find. The siblings are also highlighted in the context graph on the left (difficult to see in Fig. 6, yet visible in the tool). To reduce the complexity even further, users may choose to pull out a smaller subgraph whose root is the parent of the chosen node. This smaller subgraph is then displayed in its own tabbed panel. Similarly, a user can highlight a term node and display in a tab the subgraph whose root is that term.

**Two-way coordination of GoGet and GoView**

GoGet and GoView may be used together to enhance the exploration process. After searching for a set of terms using GoGet, users may choose a set of terms from the results and request that those terms be forwarded to GoView. GoView then displays a list of ontology subgraphs which contain that term. Each of these subgraphs may be placed in a new tabbed panel in GoView and further explored. The graph is centered on the chosen term and that term is highlighted. Conversely, after browsing the ontology subgraphs in GoView, a user may select a set of terms and choose to have the details of the gene product associations for those terms shown in a GoGet result table. Figure 7 shows GoGet results for the terms that were highlighted in Figure 1. The details of what data source a gene product came from can be displayed, and the URL for the sequence in that data source can be accessed, if available. It is only at this point that a new browser window appears—all other operations are on the same window, which helps with the burden of keeping track of open windows.

These details about the association of a gene to a term can be useful for other kinds of queries. For example, the following was also given in The Gene Ontology Consortium (2001): ‘what evidence is there that the mouse Pax6 gene product is involved in eye morphogenesis’? By choosing to search for ‘eye morphogenesis’ in the term name, for ‘Pax6’ in the Gene Product symbol name, AND between them, and restrict to the Mouse Genome Informatics data source, a user can get the single pertinent result. Then he/she can drill down to the details about what evidence supports the association of that gene to that term and discover that it is inferred from mutant phenotype and inferred from genetic interaction. The result and the query performed can all be seen on one screen in his/her browser, as shown in Figure 8. Generally, other existing GO browsers would require navigating by starting with a search in term name and searching
through a list of gene products in a second browser window to find ‘Pax6’, or by starting with a search for ‘Pax6’ in gene products and looking for its association with the ‘eye morphogenesis’ term. As the list of gene products associated with eye morphogenesis gets larger, or the number of terms that Pax6 is associated with grows, this process would slow down. Other similar questions would suffer from the same problem. Our tool provides added flexibility to obtain the answer quickly with few mouse clicks and no searching through a list by eye.

IMPLEMENTATION

Figure 9 outlines the three-tier architecture of our system. We have placed the GO data in an Oracle 8i database. The query and visualization tools have access to the database via a middleware layer of Java Objects stored in an open source J2EE application server called JBoss (http://www.jboss.org). GoGet uses JSP (Hall, 2002) served by an Apache Tomcat web server (www.apache.org). This enables us to place Java code inside our web application that obtains Java objects from the JBoss server. There are objects that request results, and other objects that hold the results that came back from Oracle. GoGet also makes extensive use of Cascading Style Sheets (CSS) and JavaScript to place all the query interface boxes and the result table seen in Figures 3–5 and 8. GoView uses some of the same middleware objects as GoGet and others specific to the graph structure view and manipulation.

Other tools with different views of the GO data could easily be added to the system, and the Java classes in the middle tier can be reused by those tools. The middle tier Java classes can be ported to any J2EE-compliant application server, because the Java code is portable across these servers. A mySQL version of the database could be used in this configuration, and only a few localized Java classes would need to be changed. However, certain aspects of the tools will be much slower, because we took advantage of some special features of Oracle 8i (explained below).

The middle tier Java classes are in separate packages that contain classes for several kinds of objects. The data access
objects, or DAO, set up the connection to the database and execute the queries requested. This is the code that would change to add a mySQL version of the database. With J2EE, a simple change to an associated XML file would change the user name and password used to connect. We have used this to switch from one version of the database to another. The Enterprise Java Beans (EJB) package contains the classes that hold session information for each client. Initializing a session EJB in a client will create the database connection through a DAO object. The queries package contains the classes for objects that contain queries that were built in the client interface. Query objects are sent through the EJB to the DAO for execution. The value objects (VO) package has classes for objects that contain a single row of data returned from a query object request. The client session EJB has a container class that holds as many of the VO objects as were returned by a particular query. The tables package contains the code for creating the HTML table that holds the results for GoGet. The views package contains client-specific classes: one class is for holding terms chosen in GoView and another is for building parts of the queries from user choices in GoGet. Developers who wish to try building new clients may obtain the J2EE middle tier Java class files from the authors. They may use our installed JBoss server or build their own. We also have developer documentation available from our website.

In addition to the database tables available from the GO Consortium, we created several additional tables and text indices that improve the speed of the queries and thus enhance the user experience. We created a large table that joined several of the original tables together, so that the join did not have to be executed each time users made requests. We sacrificed space to hold this large instantiated intermediate table for speed of execution of searches in GoGet. The middleware makes requests on the instantiated view table and returns rows in a view object for placing into HTML tables in GoGet results. We also added additional columns to the data source and evidence tables that have more explanation about what each data source is and what each evidence code means. This additional information can be seen in Figure 2, as the text beside each of the codes that are in the original GO data. The SQL for generating the intermediate view table and the additional information about the data sources and evidence codes are available upon request.
Data exploration tools for the Gene Ontology database

Fig. 8. The Macalester GoGet tool showing result of Pax6 mouse eye morphogenesis query.

Fig. 9. The three-tier system architecture.
The text searches in the GoGet tool are fast because we used a text indexing feature that is part of a package called InterMedia in Oracle 8i. There are three data fields for a term (name, definition, synonym) and three data fields for a gene product (full name, symbol, synonym) that contain text strings. We built a special index on the text in each field. Each index is stored separately and used for a ‘contains’ query on that field. Having separate indices on each field means that we can give users the freedom to search any or all of them. These indices are faster than using the ‘like’ keyword of SQL, because the rows whose fields contain the text are returned in one data access operation through the index, rather than having to search through the entire table comparing each field of each row for the text. For example, when the user searches for ‘triphosphatase’ in the term’s name, the Oracle 8i query looks like this:

```
SELECT * FROM term WHERE CONTAINS (name, '%triphosphatase%') > 0;
```

The ‘%’ signs are ‘wild card’ symbols, so that longer words containing ‘triphosphatase’ somewhere within the term’s name will also be returned. This is the default on the GoGet interface, but users can choose to do exact word searches to narrow down the number of results returned. The ‘CONTAINS’ clause returns a numeric value that indicates how well the phrase being searched for matched in the text field. We chose to return anything with a positive score, since none of these text fields is particularly long (the longest is a gene product’s full name, which is a maximum of 400 characters). Another useful feature of the Oracle text indexing is that we can indicate punctuation characters that should be considered legitimate characters in words. We built the indexes so that ‘.’ and ‘,’ were word characters, which can be really useful for biological names (e.g. 7,8-dihydro-8-oxoguanine-triphosphatase).

To build the graphs seen in GoView, we also needed to transform the original data into a format suitable for generating the graph layout. This is often necessary for effective visualization, and the process has been characterized by Chi (2000). Chi’s ‘Data State Reference Model’ has three data transformation stages between raw data and final view: (1) ‘Data Transformation’, (2) ‘Visualization Transformation’ and (3) ‘Visual Mapping Transformation’. We performed these three transformations on the GO data: (1) we obtained connected subgraphs within each of the ontologies; (2) we transformed the terms and their parent–child relationships into nodes and edges of a graph and put them into a form suitable for input to the dot graph drawing program (Gansner et al., 1993; AT&T Labs Research, 2003); and (3) we executed the dot program to obtain coordinates for the location of the nodes and edges for each ontology subgraph. The result of the last transformation is used by GoView to display the graphs. We have developed scripts to handle these transformations with minimum human intervention for each new release of the GO database.

Step 1 in the transformation process is possible to perform in Oracle directly by making use of the CONNECT BY operation, which recursively connects related rows in a table of hierarchical data. For example, here is how we generate the graph whose root, or starting point, is the molecular function ontology (term id = 2):

```
SELECT term1_id, term1_name, term2_id, term2_name, level lvl
FROM term2term_term
START WITH term1_id = 2 CONNECT BY
term1_id = PRIOR term2_id;
```

The term2term_term table is one that we build as a natural join between the term2term table and the term table from the original GO data. This gives us term names along with identifiers for all terms with a parent–child (term1-term2) relationship. We are also able to get terms at the first level below the top root term, and then build all the subgraphs from those first-level terms. If some of those subgraphs are still quite large, we try to make subgraphs of the next level of terms. The farthest that we had to go was four levels for the children of the term called Metabolism, which is the child of ‘cell growth and/or maintenance’, which is a first-level child in the biological processes ontology. Many graphs are visible at level 1, just below the root of each ontology, but others must be split out to the lower levels.

Step 2 in the transformation process was a script to put the results of the above queries into files suitable for input into the dot program. Step 3, executing the dot program for each subgraph and obtaining a layout for the graph, produces a set of files that are packaged with GoView for download. With each release of the GO database, we update the graphs. We also store the following in Oracle: a list of child terms at each level and whether they are the root of one of the visible subgraphs, or whether a child must be chosen to see a graph. These data are then used to populate the lists that are seen at the top of GoView in Figures 1 and 6. The lists dynamically change as the user chooses terms that require children to be displayed (indicated with an asterisk).

Otherwise, the corresponding graph is displayed, using the file created a priori from dot. The GoView tool uses the Grappa package (AT&T Labs Research, 2002, http://www.research.att.com/sw/tools/graphviz/download.html), which we extended, to draw the graphs in both the detail and context views. Currently, users must get a new version of GoView with its packaged graph files each time we update to a new version of the GO database. We plan to change this so that when the database has been updated, user will be prompted to download the new graph files to match the new version of the GO database.

3452
Communication between GoGet and GoView is established by using the Java messaging service (JMS) in the session EJB in JBoss. The GoGet tool has a small frame at the bottom of its page that includes a small amount of JSP code that checks for a message from the GoView tool in the message queue in the JBoss JMS. The messages are sent to the JMS queue when a user requests details about terms selected in GoView. Conversely, when a user asks to see the graphs for selected terms in GoGet results, a message is sent to the JMS queue and picked out of the queue by the GoView code, which then determines what graph(s) each term is in (using an SQL query to the database) and builds a list of graphs for the user to chose from.

We have tested the GoGet tool on a variety of web browsers. The cascading style sheet usage works properly in Mozilla and Mozilla Firebird on Linux, Windows XP, Mac OS X and SunOS 8. There are a couple of misplacements of the query boxes using Internet Explorer under Mac OS X and Windows XP. GoGet generally works using the Safari browser on Mac OS X, version 10.2, except that the box containing the English version of the query must always be kept open for it to work properly. GoView will run on any computer that has Java version 1.3 or greater installed on it. We have tested it on Mac OS X, Linux and Windows XP.

GoView provides reasonably fast user response during exploration because the graphs are packaged with the Java application. We have tested GoView from locations outside of Macalester College, and it is quite responsive. The points at which requests are made to our database through the JBoss server have reasonably short wait times. The JBoss server is designed to handle large numbers of simultaneous requests, so we expect to be able to handle a large user base for both of these tools.

**DISCUSSION**

We have two applications that introduce new interfaces for exploring the GO database and asking biologically interesting questions of this increasingly important resource. We have demonstrated this with some example queries that produce useful results in one step. The tools are part of a three-tier architecture using the latest available J2EE technology, which eases the ability to add new applications and separates the 'view' web applications from database access and 'value' objects, which are reusable. To insure the effectiveness of these tools, we transformed the original data into forms that enabled fast query processing and graph visualization.

In the future, we plan to add a mySQL version of the database to our server and test user interfaces to it, reusing the EJBs and value objects. The system could also be enhanced by providing a personalized interface, in which users could return to previous work they had done when they last visited. We plan to experiment with this by enabling users to save queries and analysis results and recall them.

GoView is undergoing some enhancements that will be made available in the future. We plan to provide more detail about each term by using color and other graphical elements to indicate user-requested information, such as how many gene products are associated with a term and from what organisms (data sources or species). We are adding a fisheye lens as a means of zooming in and seeing detail in sections of the graph (Sarkar and Brown, 1992), as well as improving the zooming and panning so that the small context view of the whole subgraph is unnecessary. We also have plans to make the tabbed graphs capable of being ‘torn off’ and placed in a separate window, so that multiple graphs could be viewed side-by-side.

**ACKNOWLEDGEMENTS**

We wish to thank Michael Dickel for useful editorial comments on this manuscript, and Nyalleng Moorosi and Mariya Sadakova for research assistance while preparing this manuscript.

**REFERENCES**


AT&T Labs Research (2002) Grappa Java GUI.


