For Online Only

Supplementary Materials

DOI: 005-1145

Software implementation

MouseTRACS was written mainly in Perl which connects via DBI to a MySQL database. MySQL is open source, fast, easy to manage, and stable. We used a Perl object library to abstract common database functions to centralize and maintain referential integrity. The Web applications run on Unix-based systems such as Linux and Mac OSX and use the Apache Web server and Perl CGI module (Stein 2005).

The various functions of MouseTRACS are divided into several components. The data viewer visualizes the test data using the Data::Table Perl module (Zhou 2004). Using the open source Java jPost applet, changes are sent immediately to the database upon changing a form (Sun 2000). Instant database updating increases responsiveness on very large forms. Only data from modified forms are submitted instead of the whole form which avoids transferring megabytes of data when only a few form elements are updated. The browser uses javascript to relay changes to the jPost applet which executes the database commands via a Perl script.

The mouse colony functionality is similarly implemented using Perl, Data::Table, Javascript, and Java jPost. Here, many mouse characteristics are updated as they are filled in and some menus are dynamically generated based on the contents of other fields.

Pedigree rendering is implemented through a system call to a modified version of the open source Madeline v0.933 pedigree drawing program (Trager 2001). The program source is modified to consolidate the rendering for special breeding cases such as intercrossed siblings and parental backcrosses. In addition, the pedigree size limit is increased to accommodate very large pedigrees and pedigree printing is spread over more pages for greater legibility. Colors for affected traits are also modified to reflect just high and low values (red and blue) instead of a separate color for each trait. A Perl script creates a script and data file for Madeline to render the pedigree in postscript. The postscript output is converted to PDF using ghostscript and opened in a new window.

Cage cards are printed by calling a Perl cgi that retrieves the cage information from the database and sends them to a Java applet that renders and prints the cards. New layouts can be generated by extending the cage card class.

The Perl cgi graphing component called Graph IT dumps data to the open source R project for statistical computing platform (http://www.r-project.org/). Depending on the graph type chosen, Graph IT will dynamically create an R script that generates a postscript graph which is converted to PDF via ghostscript and returned to the user.

Configuration

Multiple versions of MouseTRACS can be run from the same code base. Depending on the incoming URL, a configuration file then determines the functionality and to what database MouseTRACS connects. Thus, an arbitrary number of MouseTRACS-type applications can be run on a single server yet be serviced by a unified code base. For example, our current configuration supports one mouse colony for the pharmacology program and a different mouse colony for the ENU program. The databases are separate although the code base is the same. Bug fixes on one system therefore are immediately deployed to all other systems and code reconciliation is minimized.

Database schema

The mouseTRACS schema contains 32 tables that model mice, cages, requests, and test data and provide logs, genotype information, breeding limits, or allele management information. The schema was designed to add additional screens easily by inserting rows to the assay and test tables. No modification of the schema is necessary. This is a key feature because over 20+ assays comprising 300 tests have been added over the past four years. Software maintenance is simplified since display, graphing, statistics, flagging, confirming, and retesting scripts need little if any modification to accommodate the new assays. The disadvantage, however, is that complex experiments involving time courses, various treatments with various compounds and temperature changes, and multiple samples cannot be neatly modeled because a separate test has to be created for each combination of factors. We have created an adjunct database for these secondary screening followup experiments that is linked but not part of the main system. The schema for the SNP database contains tables that model SNPs, genomic locations, and DNA preparation plates that are used for genotyping.

Security

MouseTRACS calls a function to determine whether a user has access to the system. This function can be overridden, but it currently authenticates versus GNF security for a valid login status. Fine-grain permissions are recorded within the database and determine whether the user can access the system and read or write to the colony

management system or the data viewer. Administrators can edit user permissions by using a separate Web application.

Mice IDs

Unified mouse tracking imposes a single mouse identifier that is used to relate test data, genotyping data, and mouse requests to a specific animal. Data tracking on animals is critically important because the right data with the wrong animals can lead to conflicting results and errant conclusions. By enforcing the use of a single mouse identifier, data mixups as a result of arbitrary numbering can be reduced. Furthermore, the output from adjunct systems such as automated sequencers and genotyping machines can be related directly to the system if the common mouse identifier is used to mark the data.

Screening data import

Data loading is a time-consuming, error-prone process. Data files need to be in a specific format in order to be loaded correctly. Often user-entered data is incorrect, especially dates. These errors often cannot be corrected machine side, since the machines all vary and may not have a suitable interface (e.g., flow cytometer). In order to address these problems, we use the following workflow: Data files of varying formats such as tab-separated text, comma-separated text, binary, and machine output are copied from the host instrument to network storage accessible by a Web server. Users then visit a Web page that lists the files placed in the folder and validates the data file. The loading script tries to identify the type of data file and calls the appropriate parser to convert the file into a common format for another script that validates the data fields and then loads the data. The validator script checks for valid dates, valid mouse numbers, and valid data. It tries to see if the incoming data matches the data range of existing data of the same test type. Mouse ID numbers are screened for common errors such as nonsequential numbers in a sequential series, e.g., 10045, 10064, 10047, resulting from a transposition error. Errors and warnings are reported to the user for correction. This instant feedback prevents much unproductive rapport between the data producer and the data loader. Once the file passes the validator script, it is moved into a loading folder from which files are automatically loaded into the database every hour.

Data flagging

Finding outliers is one of the main goals of MouseTRACS. The manual screening of thousands of data points and subsequent requesting of retests is tedious and time-consuming. MouseTRACS will automatically flag the test outliers and reschedule the corresponding mice for retesting in the appropriate assay. Or, if the new data is retest data, then it will be compared with previous data and marked as either confirming or nonconfirming.

Upon loading new data, a script is called to create test thresholds from which flagging criteria will be drawn. Depending upon the generation, background, and gender of the mice, various methods are used to create the thresholds. In the simplest case, data for a test from mice in the initial G3 screen is gathered from a sliding window of test dates such that at least 200 non-retest data points are used to adjust for instrument drift, i.e., the tendency for baseline values to trend up and down. The data values are sorted and 1% of the values are removed from the top and bottom of the distribution. Some tests with non-normal data distributions are then log-transformed. The mean and standard deviation are calculated from this distribution. A high and a low value are calculated from a multiple of the standard deviations from the mean. The default multiple is 2 standard deviations for low and high thresholds but each can be configured independently for each test. The thresholds are stored in the stat table.

The second case occurs during heritability testing where a small but deviant population of mutants is mixed in with a majority of nonmutant littermates. To identify the mutants, successive rounds of data trimming attempt to center the median value in the middle of the nonmutant distribution. Briefly, the median and standard deviation are calculated for the whole distribution of mixed mice. Values outside of 2 standard deviations off the median are discarded. This step can be repeated several times until a single distribution is evident. The high and low values are calculated from the last median and standard deviation.

Because test values for many assays have different distributions depending on gender and background, separate statistics are calculated for each different combination. For instance, males have a slightly greater percentage of B cells than females, while females have a greater percentage of T cells. Failing to take sex into account can obscure mutant identification. Scores are normalized by recording the z-score, or number of standard deviations off the mean or median (depending upon the scheme), so that mutants can be compared relative to the proper distribution.

Once test thresholds are calculated and stored in the database, new test data is flagged if it falls outside of the thresholds. If the flagged test is for an initial screen, the mouse is automatically put on the task list for retesting. If the flagged test is the

retest result, it is compared with the initial test result. If the flags match, i.e., they are both high or both low, then the results are marked as confirmed, otherwise they are marked as not confirmed. If the retest did not flag, then the script checks if the retest result falls within 5% of the difference between the high and low thresholds from the nearest threshold. For example if a test has a low threshold of 1 and a high threshold of 100 but the test scored only 95, the test would still be marked as confirming even though it is lower than the high threshold. Thus, confirming tests that did not flag indicate borderline cases that need further examination. After data loading and analysis, completed retest requests are automatically taken off the task list. Emails are sent to researchers when mutant family lines accumulate multiple affected animals.

References

1. Stein L (2005) Simple Common Gateway Interface Class (http://search.cpan.org/dist/CGI.pm/)
2. Sun Microsystems Jpost Applet (2000) (http://java.sun.com/openstudio/jpost/index.html)
3. Trager EH (2001) open source Madeline v0.933 pedigree drawing program (http://eyegene.ophthy.med.umich.edu/madeline-0.933/index.html)
4. Zhou Y (2004) Data type related to database tables, spreadsheets, CSV/TSV files, HTML table displays, etc. (http://search.cpan.org/~ezdb/)