GENOME WIDE TRANSCRIPTIONAL COMPARISON OF BATCH AND CHEMOSTAT NUTRIENT LIMITED CULTURES OF *SACCHAROMYCES CEREVISIAE*

A DISSERTATION

SUBMITTED TO THE DEPARTMENT OF GENETICS

AND THE COMMITTEE ON GRADUATE STUDIES

OF STANFORD UNIVERSITY

IN PARTIAL FULFULLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

Alok Jerome Saldanha

September 2003

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

_____

David Botstein
Principal Advisor

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

_____

J. Michael Cherry

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

_____

Tim Stearns

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

_____

Russ Altman

Approved for the University Committee on Graduate Studies:

_____

Abstract

The chemostat, a continuous culture system, may provide a better controlled environment for detailed studies using highly sensitive, genome-wide techniques than the widely used batch culture. However, the relation and relevance of the chemostat to batch culture has not been fully characterized.

In the first chapter of this thesis I use genome-wide transcriptional profiling to compare batch timepoints and a steady-state chemostat culture under phosphate, sulfate, leucine and uracil limitation. The profiles, together with physiological data, show that phosphate and sulfate limited batch cultures maintain homeostasis as the limiting nutrient is depleted, and are comparable to the chemostat near the point when the limiting nutrient is exhausted. Importantly, there is not a stress response in the chemostat. Leucine and uracil limited chemostat cultures also appear to lack a stress response, but the existence of a comparable point is more difficult to ascertain. Comparison of changes across the timecourses revealed induction of genes involved in the metabolism of the limiting nutrient for most nutrients, but there was no relevant response found for uracil limitation. Several other clusters specific to particular limitations provide insight into the response to those limitations.

In the second chapter, I present the design and implementation of Java Treeview, a general, cross platform, open source visualization program for genome-wide microarray data.

Acknowledgements

I would like to thank my entire cohort of Botstein Lab yeast people, starting with Jon Binkley and Ellie Click, Kirk Anders, Matt Brauer, Maitreya Dunham and later, Michael Shapria, Jagoree Roy and Michal Ronen. I have learned something valuable from every one of you, and together we had the best lab environment one could hope for. I would also like to thank the human side of the lab, particularly John Murray, Mike Whitfield and Mitch Garber, who were my home away from home in the final days. Rob “Gets” Pesich was consistently the solution to any reagent supply problems, and every lab needs someone as well organized and thoughtful. Katja Schwartz is a master of yeast and the ultimate source of any technical proficiency I have. With the exception of microscopy, in which I received valuable training from Jon Mulholland. I would like to thank Dr. Dunham for her friendship, interesting discussions, good times, and also for teaching me basic techniques when I was too ashamed to ask anyone else.

I would like to thank my good friend Mr. Raymond Luk for good times and perspective when the going got tough, and Miss Carole Lu, who got tough when I lost perspective.

David Botstein has been an excellent advisor, helping me see the way forward amid many distractions. Thank you for allowing me the freedom to pursue my ideas, while providing enough guidance to ensure I make progress.

Thanks also to Pat Brown, Julie Baker, Arend Sidow, and Man-wah Tan, who have all given me valuable advice on several occasions, and to Gerry Fink and Boris Magasanik, my advisors at MIT, and to Hiten Madhani and Tim Galitski, who got me into this business in the first place.

My thesis work was supported by a National Defense Science and Engineering Grant, and also by the Genome Training Grant.

Table of Contents

Table of Tables

Table of Equations

Table of Code Listings

Table of Figures

Introduction

This is truly an exciting time to be a biologist. The availability of genome sequence, the informatics resources to utilize it, and high throughput assays to generate more data hold the promise of a more comprehensive understanding of many aspects of biology. However, the comprehensive nature of whole-genome techniques raise new issues; subtle changes in experimental conditions can lead to complicating artifacts in the data.

The first chapter of this thesis is dedicated to the study of the transcriptional programs in batch and chemostat cultures of yeast. The relevant issue is how the transcriptional network responds to prolonged limitation for a particular nutrient; does the culture enter a state inaccessible to batch grown cells, is it undergoing a persistent stress response due to starvation, or does it maintain a homeostatic balance?

In addition to addressing the questions of the relationship of the batch to the chemostat, these data also allow me to characterize the transcriptional response to running out of the limiting nutrient during the timecourse. This gives insight into the physiological response particular to each limitation.

In the second chapter, I present Java Treeview, a general tool for visualization of microarray data. The value of large datasets is limited by our ability to make sense of them. Java Treeview provides the proven visualization of hierarchical clustering in a scalable, platform-independent package. It also

includes two new displays: a scatterplot display for visualizing the relationship between per-gene statistics and the clustering, and a karyoscope display, that visualizes the gene expression data in genome order. All of these displays are customizable and support export. Java Treeview is also extensible, open-source, and freely available from http://jtreeview.sourceforge.net. Indeed, in the first 40 days since the website was set up Java Treeview has been downloaded over 300 times, and several people have contacted me with proposed extensions.

Chapter 1

A Comparison of Batch and Chemostat Cultures of *Saccharomyces cerevisiae* under Phosphate, Sulfate, Leucine and Uracil Limitation.

Introduction

Metabolism is flexible. Common laboratory organisms can consequently be grown on a wide variety of media. However, there are certain classes of nutrient of which at least one member must be present. There must be a source for each of the elements which comprise the organism, including carbon, nitrogen, phosphorus, sulfur and other less abundant salts and minerals. For non-photosynthetic organisms, there must also be an energy

Chapter 2

Java Treeview

Introduction

With the advent of whole-genome analysis techniques, the collection of hundreds of thousands of measurements by a single researcher has become routine. The sheer volume of data collected makes it impractical to consider all of the data values manually. Thus, the statistical analysis of data has become a topic of major importance to genome biologists. By characterizing genes and experiments statistically, one can avoid having to consider each one individually. Simple patterns and groups can be extracted from the sea of data which can then be compared with biologically relevant factors.

A key companion to the generation of appropriate statistical measures is the ability to visualize them. One of the key ideas which led to the long term success of Michael Eisen's Treeview program was the separation between the generation of the clustergram, which can be achieved by multiple different programs, and the display, which is handled entirely by Treeview. Thus, it suddenly become possible to generate clustergrams according to arbitrary metrics, and then visualize them with the same program. The key benefit to the developer was that they no longer needed to worry about how they would

display the data. The key benefit to the user was that they could view the results of novel analysis from the same interface as before.

Java Treeview takes this approach another step. It extends the file format of Treeview to allow the incorporation of more types of data, and adds several new displays to view the new types. Finally, it links these displays together with a unified selection model to facilitate comparison between the types.

Overview of Original Treeview

Java Treeview is heavily based upon the original Treeview by Michael Eisen (Eisen, Spellman et al. 1998), which is in many ways simpler. It is therefore helpful to have an understanding of the purpose and operation of the original Treeview before trying to understand how Java Treeview works.

Treeview displays the result of hierarchically clustered gene expression data in a way that is convenient and intuitive to navigate. One of the biggest discoveries enabled by whole genome microarray technology is the coordinated regulation of whole pathways corresponding to biological processes. It is this aspect which has made the technique so useful, and which hierarchical clustering identifies so well.

Treeview generates its display from three simple tab-delimited text files. The first, the cdt file, contains the clustered expression data along with some gene identifiers. There is one gene per row, and one array per column. The

ATR and GTR files contain a flattened form of the hierarchical tree, with one row per node, one column each for left and right child, and one column for the correlation.

The display produced by Treeview allows one to navigate the gene and array trees, and click on the image of the expression data to produce a zoomed- in image of the selected genes in an adjacent panel. However, it has many limitations, and is coded in an obscure toolkit so that it is annoying to develop.

The limitations which Java Treeview seeks to overcome are the platform dependence, which limits users, and the toolkit, which limits developers. It also seeks to have excellent performance on large datasets, and to be deployable under the widest set of circumstances, including java applets.

Java Treeview is a complex program consisting of over 42,000 lines of code. It is still under development, and the extent to which it will be carried is unknown. There are groups which are making it into a Java Web Start application for viewing a web database, a group which has added support for visualization of a novel clustering algorithm, and SMD at Stanford has expressed interest in an Applet version for viewing the output of SMD's clustering. In truth, all these things are possible and more if there is interest. It is difficult to summarize the amount of effort which has gone into the development. This chapter will consist of two major sections. The first is a

description of the architecture, focusing on key design decisions and the reasons behind them. The second will be a walkthrough of how the different parts of Java Treeview work, to illustrate how the architecture comes together in a working application.

Design of Java Treeview

Choice of Platform: Java

A major drawback of the original Eisen Treeview is that it is written using an obscure toolkit and consequently only runs on a single platform. This hinders usability for end users, who may have more convenient access to an alternative platform, and places an even higher barrier for the developer, who even if he has access to the platform, will most likely not be familiar with the toolkit. These drawbacks seemed difficult to circumvent until the advent of the Java Programming Language (Joy, Steele et al. 2000). With Java, it is now possible to write robust, scalable applications which deliver rich interfaces in a cross platform manner from a single source code base. Additionally, Java applications can be easily modified to form Applets, if care has been taken to stay within the limitations that browsers impose. To maximize the pool of developers, and users, and hence the utility of the resulting program, I chose

to write rewrite and extend Treeview in Java rather than extend the version originally written by Eisen.

Persistence of Configuration Information: The ConfigNode Interface

One of the things which impacts the usability of a piece of computer software is the amount of time and effort it takes to get it into a state where useful work can be done. Ideally, double-clicking the icon would open exactly to the file you want to view with all of the appropriate options set. At the other extreme, the software could start in the same default state every time you open it, and force you to hunt for files and configure colors. The business of storing configuration information is an attempt to make the user experience more like the former.

Because Java Treeview is designed to be extensible, it did not make sense to store configuration information in a central location. A view might be instantiated more than once for a particular dataset; if the two views had different settings and stored them in the same place, they would clobber each other. Furthermore, since different views may be developed at different times by different developers, there is a chance that they would again clobber each other's values.

A solution to this issue is to use a hierarchical data structure. Each object can be bound to a node of the data structure. The node supports the

ability to store key-value pairs with specified default values, and also the ability to create subnodes. Thus, an object which contains another object can simply allocate a subnode for it, and then bind the “sub-object” to the subnode. The interface I chose to implement this idea, ConfigNode, is listed in Code Listing 2-1. There are currently two concrete instantiations of this class, one of which is an inner class of XmlConfig and represents a node in an XML document, and the other of which is DummyConfigNode, which is not bound to persistent storage, and can be used for testing as well as when persistence is not desired.

This method of storing configuration information is used both to store per-document settings as well as program-wide presets. The complete configuration graph for a typical document’s settings is displayed in Figure 2-1. I have tried to decompose the graph into comprehensible chunks by grouping together nodes which represent configuration of similar things. The corresponding Xml document is listed in Code Listing 2-2. The complexity of the state makes it difficult to maintain without a hierarchical data structure.

Decoupling Views and Models: DataMatrix, HeaderInfo and DataModel

Within the application, there are many components which must get data from the data model in order to do their job. Some of these components are graphical in nature, and must access data values, as well as recognize which

values represent missing data. Others depend only on certain types of annotation to the rows and columns. However, if an explicit dependence on the TVModel class is hard-coded into the View classes, then all other models will be forced to extend TVModel. If the other model does not support some of the features of TVModel, it will be forced to stub out the functionality to prevent unexpected behavior. Furthermore, there are some “views”, such as UrlExtractor and HeaderSummary, which depend upon attributes of rows and columns which are completely symmetric. If the row instantiation and the column instantiation both maintain references to the same TVModel, they will need to retain knowledge of which orientation they are. This has the disadvantage of reducing their reusability in the context of other models, if there is a third type of header. These tensions can be resolved with the introduction of interfaces which can be created in a variety of ways by data models, and used to flexibly reuse View classes on different aspects of the same data model. The three interfaces are the DataMatrix interface, the HeaderInfo interface, and the DataModel interface, which are listed in Code Listing 2-3, Code Listing 2-4 and Code Listing 2-5.

These interfaces greatly increase the reusability of Views. For example, the same UrlExtractor class is used to construct URLs for genes and arrays. The constructor for UrlExtractor accepts a HeaderInfo object, but does not know or care whether that object represents gene headers or array headers. The URL template and the appropriate header to insert into the template are

stored in its ConfigNode. When it is passed in an index, it simply uses the index to look up the appropriate headers in the HeaderInfo and fill out the template.

I anticipate as the variety of analysis methods viewable by Java Treeview expands, additional interfaces may be added. For instance, it is likely that an interface representing a tree will have to be introduced once there are more data models which produce trees, and the data associated with the trees becomes richer.

Graphics Performance: Tree Traversal and Pixel Buffering

Graphics performance under Java can be problematic. The main issue is the high level of abstraction at which drawing is usually handled. Although this allows device independent drawing in a natural way, it also incurs a performance penalty.

There are three particular types of views in Java Treeview which suffer substantially from the drawing performance of Java. The first is the gene tree, drawn by LeftTreeDrawer on GTRView. There are simply too many lines which must be drawn if the entire tree is to be displayed. The second includes the pixel views, the GlobalView, which shows a zoomed out overview of the data, and the ZoomView, which shows the zoomed in view of the selected data. The third is the ArrayNameView.

To speed up drawing of the gene tree, a variety of strategies which avoid unnecessary redrawing are used. In order to exploit these strategies, Java Treeview maintains an offscreen buffer of currently visible part of the tree (buffering the entire tree would take prohibitively large amounts of memory). This has the additional benefit that whenever the Swing graphics subsystem requests a portion of the image, it can be quickly copied. Since this strategy is of benefit to other views, it is implemented by ModelViewBuffered, the superclass of GTRView. The first trick to rapidly drawing the gene tree is the observation that if the leftmost and rightmost children of a subtree are both above or below the visible portion of the gene tree, the entire subtree can be skipped. This allows for a huge speedup in tree drawing when scrolling a zoomed-in GlobalView. Furthermore, during initial image generation as the LeftTreeDrawer traverses the tree, it checks to ensure that at least some member of a child is visible before recursively drawing it. The second trick is to recognize that when a subtree is selected, the rest of the tree does not need to be redrawn. This principle is used in several ways. First, when the mouse is clicked, instead of redrawing the whole tree Java Treeview first draws over the currently selected subtree in the non-selected color, followed by drawing the newly selected subtree in the selected color. Secondly, when a parent node is selected, we merely need to redraw the parent node and its formerly unselected children in the selected color. The reverse is true when a child of the currently selected node is selected.

A different strategy entirely is required to speed up drawing of the GlobalView and ZoomView by the PixelDrawer. The key issue is the sheer number of colors which appear in these views. Using the standard java Graphics interface, the only way to draw multiple colors is to create a Color object for each one. This leads to unacceptable overhead, particularly on Mac OSX. A solution to this problem is to use the MemoryImageSource interface, which allows one to create an image which displays pixel data stored in an array of int. Furthermore, changing the values in the int array does not trigger an update until MemoryImageSource.newPixels() is called. This is not quite as good as directly manipulating a pixel buffer, but in practice it is sufficiently fast.

A final difficulty is drawing vertical text. Previous to Graphics2D, Java had no way of drawing rotated text. Thus, the only way to generate a view with vertical text is to draw it horizontally to an offscreen image and then rotate it. Because wide compatibility was a major design constraint in Java Treeview, this was the strategy pursued. However, rotating an offscreen buffer is not as fast as one would hope. It is likely that additional speed could be gained by avoiding the use of image rotation when Graphics2D is available.

An Extensible File Format: The Generalized CDT File

It is difficult to forsee what types of data will be produced and how best to manage them. For instance, we may want to record per-gene statistics or category information, or perhaps specify different colors by which the names

are rendered. In order to do all of these things, we need a very flexible file format which has enough structure that applications can associate data with the appropriate items, but is general enough that the format will not need to be rewritten every time we want to include a new type of data.

The Generalized CDT (GCDT) file format is a straightforward generalization of the PCL and CDT file formats used by common clustering tools which fulfills these requirements. This file format is a table formed of tab-delimited text with a few special extra constraints which make it well suited for the kind of data generated by gene expression studies – it requires the inclusion of a special EWEIGHT row and GWEIGHT column. The idea is that the lower right hand corner of the tab-delimited file is purely microarray data. In addition to microarray data, this file can contain additional per-gene and per-array annotation in columns before the GWEIGHT column or in rows before the EWEIGHT row. The one additional detail is there is a special annotation column, called the unique identifier column, which is either the first column in the file, or the second if the first column is named “GID”. This is because in traditional CDT files the node identifiers are placed in the first column, which is labeled GID by convention, pushing the actual unique identifiers into the second column. Also for backwards compatibility with the earlier PCL and CDT formats, if the GWEIGHT column is missing Java TreeView assumes the data starts on the third column, or the fourth column if

the first column has the header GID. Similarly, if the EWEIGHT row is missing Java TreeView assumes the data starts on the second row.

Using the Generalized CDT format, additional per-gene and per-array annotations and scores can be added to the file and carried through multiple steps of analysis. Applications can use the column and row headers to decide whether a particular annotation is meaningful to them, and ignore the rest. They can also present the column names to the user for configuration of various analyses. This file format is support by the PCL_Analysis Perl package (http://pcl-analysis.sourceforge.net) which is highly recommended for use with Java Treeview.

It should be mentioned that there are two other file formats, the tree files, which end in a .gtr or .cdt extension, and the xml-format configuration files, which end in a .jtv extension. The tree files are tab-delimitted text with exactly four columns corresponding to node id, left child, right child, and correlation. The XML configuration file is a straightforward representation of the tree of ConfigNodes. For further details on the XML format, see the XmlConfig.java source file.

Enabling Modularity: Generic Structure of Views

If a single application is going to incorporate an unspecified number of views of unknown type, it is essential that they conform to a particular interface through which the application can manage them. This is yet another instance

of a concept which was applied liberally throughout Java Treeview, as evidenced by the earlier discussion about HeaderInfo, DataMatrix and DataModel.

All data displays in Java Treeview are implemented in a stereotyped fashion. The classes which represent a display of the data implement an interface named MainPanel. This nomenclature borrows from the Java class JPanel, which represents a panel within a window. The MainPanel generally will contain one or more ModelViews, which actually draw the images. The MainPanel is generally constructed around a DataModel, and may require additional configuration, for instance what to plot in a scatterplot. The ModelViews typically do not have explicit dependency on any model and are instead constructed using a HeaderInfo or DataMatrix. The MainPanel is generally cognizant of the ModelViews which it features, and can populate a pulldown menu with items that allow their configuration. The source code for the MainPanel interface is listed in Code Listing 2-6, and specifies exactly the methods an implementing class must provide. As is clear from the source, a class implementing MainPanel must provide methods to populate various menus, must be able to save and restore itself from a ConfigNode, and must be able to scroll itself so that a particular gene index is visible.

The MainPanel itself must be displayed in a frame. The ViewFrame abstract class describes the common functionality that a MainPanel can expect from its containing frame. The source code for this abstract class is

listed in Code Listing 2-8. A ViewFrame must be able to observe and coordinate MainPanels if necessary. Currently, a ViewFrame must also provide access to shared data structures, such as the selection model, implemented by CdtSelection, the various presets and the shared data model, typically an implementer of DataModel. Provision of this data to the MainPanels may shift to the constructors to reduce the dependency of the MainPanels on the ViewFrame abstract class. The ViewFrame abstract class also provides some functionality which is required by all extending classes, such as management of the most recently used list of files (FileMRU), initial window placement and tracking, and the ability to open urls. With the addition of the ViewFrame, the complete operation of Java Treeview can be understood at an abstract level. The main() routine sets up an application, either TreeViewApp, or its subclasses LinkedViewApp and KmeansViewApp, which creates a tree of ConfigNodes from a global presets file. The application then opens a ViewFrame and waits for user interaction. When the user indicates a file is to be opened, a DataModel representing that file is created, and a tree of ConfigNodes representing the file state is created. According to the ConfigNodes , the ViewFrame constructs the appropriate MainPanels, which then again consult the ConfigNodes to construct the appropriate ModelViews. All changes to state are stored in the ConfigNodes, and for persistence the entire tree is written out on exit, or when another file is loaded.

Enabling Shared Selection: The CdtSelection object

A key goal of Java Treeview was not only to allow the creation of multiple displays of the data, but to make it easy to see how they relate to each other. One step towards this goal is to have a shared selection model. With this in place, when a gene or array is selected in one view, it is selected in all views. Already something like this is required for coordination within the Dendrogram display from the Eisen Treeview between the Global Pixels and the Zoom Pixels. By making the selection model program wide, we can reap similar benefits for other displays of the data.

The CdtSelection object is the means by which this was implemented. It is maintained by the ViewFrame, and hence is document-wide, although not program-wide like the presets and recently used file list.

Implementation of Java Treeview

In this section, the various displays in Java TreeView are discussed in detail. For each display the contents and possible manipulations, the creation and configuration by the user, implementation in java objects, construction by the application and export options are described. The shared code for interfaces is kept in the main source directory, whereas the code for each display is kept in a separate subdirectory to minimize dependencies. The code for the applications is kept in a separate “app” subdirectory. With the

introduction of the DataModel interface, it should be possible to additionally move all concrete implementations in to a special “model” subdirectory in the future to enforce decomposition.

At this point, it must be noted that Java Treeview was initially developed as a framework. The idea was to provide a set of views and functionality out of which users could easily build one-off applications and test out novel ideas. This was intentionally done to avoid contamination of the main Treeview application with code of dubious quality from other contributors, while not restricting the utility of the code. Over time, it has become apparent that the proliferation of Java Treeview associated applications can be confusing to the user as well as novice developers. The model towards which Java Treeview is moving is a single application which can load files as different types. Under this model, the user will select a file, and simultaneously select the display type from a pulldown menu labeled “Display as” within the same dialog. The initial display types will be “Autodetect”, which causes Java Treeview to attempt to detect the appropriate display automatically, “Treeview” for the classic Treeview interface, “Linkedview” if additional linked visualizations are desired, and “K-means”, if the file was produced using K-means clustering. The type will determine which data model the data is loaded into, and which interface is presented to the user. For convenience, the last selected type will be stored in the global presets file, and the type of previously loaded files will be stored in the most recently used list along with the file name. This will make

it convenient to reopen files with the previous display type, but possible to reopen them with another (by using the “File->Open…” menu item).

This user interface has not yet been implemented, so these examples will be described from the perspective of the currently working Linkedview application, which is the most general of the three.

Description of ViewFrame Functionality

This section describes functionality that is provided by the ViewFrame abstract class, and hence available to all of the MainPanels. The primary functions are file management under the File menu , management of presets for the various views under the Settings menu, searching the dataset for genes with annotation matching a pattern under the Analysis menu, export of tab-delimited text under the Export menu, and window management under the Window menu.

File Management

The ViewFrame provides file management. File selection is initiated by the user selecting “File->Open”, and is provided by the standard swing JFileChooser dialog, shown in Figure 2-2 part E. As was mentioned before, this dialog will be modified to accommodate a pulldown for display type. There is a submenu on the File menu, “File->Recent Files”, which contains a list of most recently used files from which the user can select. The last file related item is “File->Edit Recent Files…”, which displays the “Edit File List” dialog

depicted in Figure 2-2 part F. The final item on the File menu is “File->Exit”, which simply closes the application as one might expect.

Within the “Edit File List” dialog, the user may update the locations of files they may have moved by selecting the file and pressing the “Find” button, and remove files from the list by selecting a set of files and pressing the “Remove” button. Another useful feature is the ability to clear the file list by pressing the “Remove All” button.

Preset Management

Presets, along with settings, comprise the means by which Java Treeview is configured. Because presets potentially apply to all the displays, the ViewFrame provides management of them, whereas the settings are managed by the displays themselves. The presets are configured using the items in the Presets submenu of the Settings menu. All of the menu items open up the “Presets” tabbed configuration panel. Although “Presets” currently has 6 tabs, there are three primary types of presets.

Presets specify a commonly used configuration of settings. The user also may also designate a particular preset to be the default. The default preset specifies the settings which are used for a file which does not have document-level settings associated with it. The presets can be edited when no files are loaded whereas the document settings, being associated with a file, can only be edited when a file is opened. It is important to remember that

presets specify common settings, and although the default preset is used as the settings for a naïve file, it can be overridden by changing the settings for the file directly. Referring back to the section on ConfigNodes, the ConfigNodes for presets are stored in the system-wide settings file, whereas those of the settings are stored in the document-level .jtv file. The distinction will become clear when settings are discussed.

The first type of preset consists of the url presets, Gene Url Presets and Array Url Presets. The dialog for the Gene Url Presets is depicted in Figure 2-2 part A. The user may change the name of one of the presets, edit the URL template which it links to, and select one to be the default. There is also a special template called None, which can be selected if there is no appropriate database. This template is commonly set to be the default for the array url presets, as there is generally not an extensive online database for array information.

The second major type of preset is comprised of the color presets, “DendroColor”, “ScatterColor” and “KaryoColor”. There is a separate type of color preset for each type of MainPanel because each display uses a different numbers of colors, and uses them in ways which do not easily map from one display to the other. The color presets panel for the Karyoscope display is shown in Figure 2-2 part B. The name of the preset is listed in the first column, and can be edited. The colors for that preset are shown in the next column,

followed by a Remove button which removes the preset, and a radio button which allows one of the presets to be designated the default preset.

Clicking on one of the colors will pop up a standard color selection dialog, shown in Figure 2-2 part H. The dialog title indicates the color being edited, and the user may choose from various color swatches, HSB color values, and RGB color values. The user may also add a new color set by pressing the “Add New” button, and add standard colors which were present the first time Java Treeview was run by pressing the “Add Standards” button in case they have removed them.

The final kind of preset concerns the genomic locations of the different loci. The interface, depicted in Figure 2-2 part C, allows the user to rename a coordinates preset, search for it using the “Find…” button, remove it by pressing the “Remove” button, and set it to be default with the “Default?” radio buttons. The “Find…” button opens a standard file chooser dialog, similar to that in Figure 2-2 part E.

The coordinates file is actually a standard Generalized CDT file with a few specialized annotation columns, as suggested by the name “YeastCoordinates.pcl” in the figure. The required columns will be discussed in more detail in the section “Implementation of the Karyoscope display”. When a Karyoscope display is created, the file indicated by the default preset is parsed and linked up with the current data model using the unique identifier column described in the Generalized CDT specification. If the default preset is set to

None, then the data model itself is searched for the special columns. This will be described in more detail in the section on the Karyoscope display.

Searching for Genes

The ViewFrame also provides the ability to search for genes. Selecting "Analysis->Find…" displays the "Search Gene Text for Substring" dialog, shown in Figure 2-2 part G. The user will generally enter text in the "Enter Substring" text field, click the "Search" button to search for the substring, and then select one or more genes from the result list. Because CdtSelection supports discontinuous selection, this does not pose a problem. In addition, the user can specify a case sensitive search using the "Case Sensitive" checkbox, select the next gene in the list after the currently selected one using the "Next" button and select all genes using the "All" button. The last button, "Summary Popup" displays a popup window similar to the ZoomView of the Dendrogram display depicting the gene expression and annotation of just the selected genes. This button only works if a Dendrogram display is currently selected, for reasons that will become clear in the later section on the Summary display.

The reason that the ViewFrame is able to provide gene searching functionality independent of the MainPanel is primarly because of the scrollToIndex(int) method provided by the MainPanel interface. By combining this method with the shared DataModel and the shared selection model

implemented by CdtSelection, the ViewFrame itself can search for genes, select genes, and scroll the MainPanels to the appropriate location.

Tab-delimited Text Export

The shared selection model and data model makes it practical for the ViewFrame to directly support export of subsets of the data to tab-delimited text. Selecting “Export->Export to Text File…” when some set of genes is selected will cause the “Gene List Maker” dialog shown in Figure 2-2 part D to be displayed. This dialog is somewhat misleadingly named, as it supports highly configurable export of the data. By default, only the unique ids are printed; since there is only one value in each row, no tabs are added, and this has the effect of making a gene list. The user can select any subset of the annotation headers using the “Field(s) to print:” list, and optionally include the expression data and a header line using the “Expression Data?” and “Header Line?” check boxes. This flexibility allows easy export of subsets of the data into other applications such as Excel. Again, it only exports data for the currently selected data, which can be selected in a variety of ways through any of the views.

Window Management

Java Treeview applications can have multiple windows open. The code which manages this is spread between the ViewFrame abstract class and the actual application class. This is because there needs to be a central list of

open windows, which clearly cannot be maintained by the ViewFrame. However, currently the ViewFrame must communicate with the application class through its concrete subclass, since there is no generic interface for application classes. This suggests the introduction of an App interface at some point in the future.

The “Window” menu consists of a list of current windows and their associated keyboard shortcuts, which are simply the apple key together with the window’s number, followed by the “New Window” and “Close Window” items. Selecting the window name, or typing the keyboard shortcut will cause the window to be moved to the top. Each window in Java Treeview is an instance of a ViewFrame subclass. Selecting the “New Window” item creates a new ViewFrame instance, and selecting the “Close Window” item closes the current ViewFrame instance.

Description of Dendrogram Display

The Dendrogram display is based upon the interface of the original Treeview by Eisen. The display is pictured in Figure 2-3. This is the primary view used in Linkedview, and is familiar to many researchers.

Dendrogram Functionality

The Dendrogram display is extremely useful because it presents the data at two different scales. On the left, the global pixels, gene tree and array tree present a zoomed out view of the data in which broad patterns of gene

expression can be observed. From this view, the user can select a subset of the data by clicking and dragging on the global pixels, or by clicking on the gene tree. If a node in the array tree is clicked, or if the shift key is held down on the global pixels, arrays can be selected as well. On the right, the array names, zoom array tree, zoom pixels and gene annotation provide details on the selected genes. Clicking on a gene annotation or an array name opens a link to a web database with additional information. The boundaries between any of the components can be moved simply by clicking and dragging with the cursor. There is also a special menu item available only when a Dendrogram display has the focus, “Analysis->Create Summary…”, which will be discussed later in the section on the Summary Display.

Keyboard input in Java Treeview is sent to the component containing the cursor. If the cursor is over the Global Pixels, the selected area can be moved around using the arrow keys, and grown and shrunk by holding the control key and pressing the arrow keys. If the cursor is over the Gene Tree, Array Tree or Zoom Array Tree, the arrow keys select the parents and children of the currently selected node.

The Dendrogram view also provides information and hints in a variety of ways. Letting the cursor linger on the Zoom Pixels causes the value represented by the pixel to be displayed in a ToolTip. This information, as well as the row, column, array name and gene annotation for that pixel is displayed in the Status Panel. The contents of the Status Panel change to reflect the

status of the view which currently has focus. Thus, moving the cursor to the Global Pixels will cause the number and extent of genes and arrays in the current selection to be displayed. Moving the cursor over the Gene Tree, Array Tree or Zoom Array Tree will cause the identity and correlation of the currently selected node to be displayed. Moving the cursor to any of the displays causes usage information for that panel to be displayed in the Hints Panel.

Recognized Annotation

The Dendrogram display specifically looks for two types of annotation, the FGCOLOR row and the FGCOLOR column. When it finds these annotations, it attempts to color in the gene annotation using the values in the FGCOLOR column, and the array annotations using the FGCOLOR row. The colors must be specified in standard hex notation, #RRGGBB.

Dendrogram Configuration

The Dendrogram display allows the user to set the color and pixel spacing of the global and zoom views, the gene and array url linking, and the fonts of the gene annotation and array names. All of this functionality is organized topically into three dialogs which can be accessed from the Settings menu when a Dendrogram display is selected.

Selecting “Settings->Pixel Settings…” will display the dialog shown in Figure 2-4 part A. The first two sections of this complicated dialog deal with the global and zoom X and Y pixel settings. This determines the number of

pixels allocated to each row and column of the data file in the global pixels and zoom pixels views. It can either be set to a particular real value, or set to fill the available pixels. In the event that fewer than one pixel is allocated to a row or column, the fractional pixel value for that row or column is truncated, the row or column is averaged together with others that map to the same pixel, and the averaged value is displayed. The next section of the panel deals with the contrast, which can either be typed in or set interactively with the scrollbar. The contrast setting determines what data value corresponds to the most intense up or down color. Values of greater magnitude than this will be displayed with the same color, and values of lesser magnitude will be displayed with a linear interpolation of the up and zero, or zero and down colors in RGB space. The final section deals with setting the actual colors. The color values themselves are displayed in the top row, followed by a row of buttons dealing with loading and storing color sets, followed by a row of buttons corresponding to exising presets. Clicking on the color values themselves will bring up a color selection dialog similar to that in Figure 2-2 part H. The “Load…” and “Store…” buttons bring up load and save dialogs similar to those in Figure 2-2 part E which allow the user to save and retrieve color information to the color file format used by Eisen’s Treeview. The third button, “Make Preset”, creates a preset out of the current color settings and adds it to the program-wide list. The last row of buttons in this section corresponds to the existing presets, and clicking any one of them updates the

colors to reflect that preset. This provides a convenient mechanism for switching between common color settings.

Selecting “Settings->Font Settings…” causes the “Font Settings” dialog depicted in Figure 2-4 part B to be displayed. This dialog allows the configuration of font face and style using pulldown menus, as well point size via a text field. There is also a preview region where the user can see how a common bit of text is rendered by the font settings. By selecting the different tabbed panels, the user can set fonts for either genes or arrays.

Finally, the “Url Settings” dialog is displayed by selecting “Settings->Url Settings…”. This dialog, depicted in Figure 2-4 part C, allows direct editing of the URL template as well as selection the annotation to be used to fill out the template. The user may also choose a template from the presets listed in the second row, or disable URL linking entirely by unchecking the “Enable” checkbox. Using the tabs at top of the dialog, the user can set linking for both genes and arrays.

This completes the description of the user interfaces by which the Dendrogram display can be configured, but does not describe the actual program structure which allows these choices to ultimately create the images. This is discussed in the next section on implementation.

Dendrogram Implementation

The Dendrogram display is implemented by the DendroView container and the components which it contains. The Java class of each of the views is indicated in parenthesis below the conventional name in Figure 2-3. The DendroView maintains references to the ViewFrame and DataModel from which it was created, and uses these to construct each of the components. When it is bound to a ConfigNode by the ViewFrame, it then binds all of its children to subnodes of that ConfigNode. The DendroView itself only manages the boundaries between the components, which are stored in the document-level xml file, and is not directly affected by any configuration other than dragging of the boundaries. It also maintain references to various shared utility objects, including instances of ArrayDrawer, UrlExtractor, GeneSummary, ColorExtractor, and MapContainer, which it provides to the various View objects. The DendroView populates the “Settings” menu with the appropriate items to configure these objects. The Views themselves listen for state changes on these utility objects and repaint themselves. Since they share instances of these objects, changing the single instance will cause all the relevant Views to update. Each view also maintains a link to the MessagePanel instances which manage the status panel and hint panel, and update them upon cursor entry.

The GlobalView and ZoomView classes, which implement the global pixels and zoom pixels respectively, delegate the mapping of rows and

columns to separate instances of the MapContainer class, drawing to an instance of the ArrayDrawer class, and color management to an instance of the ColorExtractor class. They extend ProducedModelView, and thus are produced from a int buffer, which they manage the size of, but send off to ArrayDrawer to update when there is some kind of state change. The GlobalView and ZoomView classes themselves are mainly middlemen, keeping track of user actions, display sizes and buffer management while delegating the details of what and how to draw to other classes. The only graphical elements directly drawn by the GlobalView are the selection rectangle, a yellow rectangle denoting the selected genes, and the zoom rectangle, a blue rectangle denoting the area visible in the ZoomView. These are painted on after the image buffer is drawn on the window.

To ensure that the gene and array names line up, the TextView and ArrayNameView classes, which draw the gene annotation and array names respectively, delegate their placement to the same MapContainer instances as the ZoomView. The actual content to display for each gene or array is delegated to a HeaderSummary object. Thus, these components simply keep track of the size and location of the drawable area, listen for updates, and then draw what the HeaderSummary asks them at the location specified by the MapContainer.

The GTRView, ATRView, and ATRZoomView, which draw the gene tree, array tree, and zoom array tree respectively, likewise rely upon the same

MapContainers as the GlobalView or ZoomView. Because their drawing is more complex, they also rely upon a TreeDrawer subclass, either the LeftTreeDrawer for the GTRView or the InvertedTreeDrawer for the ATRView and ATRZoomView. The TreeDrawer class itself builds up the required TreeDrawerNode data structure from data provided from the DataModel. There are really no settings that directly affect the various tree-drawing views, as they are entirely specified by the CdtSelection, window placement, and the pixel views which they adorn. It should be mentioned that the TreeDrawerNode class has provisions to store color information, which have not yet been exploited.

Dendrogram Creation

The Dendrogram display is automatically displayed when a new file is opened. Additional Dendrogram displays with independent settings can be added using the “Analysis->Make Dendrogram” menu item. When a new Dendrogram display is created, either by opening a file or due to user interaction, a DendroView object is instantiated and bound to a ConfigNode, and then added as a tab to the LinkedViewFrame. If the file had been previously opened, the DendroView is bound to the old ConfigNode to restore the state. Otherwise, a subnode of the Views node is allocated and bound to the DendroView.

Dendrogram Export

An important feature of the Dendrogram view is the ability to export images to both the vector-based postscript format as well as the pixel-based GIF format. Selecting “Export->Export Colorbar to Postscript”, “Export->Export Colorbar to Gif” and “Export->Export to Postscript” opens the dialogs depicted in Figure 2-4 parts D, E and F respectively. The fourth option, “Export->Export to Gif”, is very similar to the “Export->Export to Postscript” dialog. All export dialogs allow designation of the output file using the “Browse” button, which again opens up a file dialog similar to that depicted in Figure 2-2 part E. This is one of several commonalities in the export dialogs which simplify the task of developing export dialogs for the developer as well as their use by users. The export dialog will be described in detail here, and then covered more briefly for subsequent displays.

The “Export ColorBar” dialogs create an image of the color scale used to color in the various pixel views. They use the shared ColorExtractor and a temporary ArrayDrawer to directly color in an image buffer according to a temporary DataMatrix. The values in the DataMatrix are calculated using the current contrast settings so as to look like the color bar. The interface to configure this operation consists of a column of settings on the left, a preview panel on the right, and an output file designation on the bottom. The common options between both color bar export dialogs are the orientation of the color bar, the number of boxes, the number of decimals, and the pixel sizes. For the

postscript export only, a bounding box can optionally be included, and the size specified. A suggested size for the bounding box, and the total size of the output image, are computed automatically. The bounding box indicates the extent of the image, and is required by some postscript interpreters.

The “Export to Postscript” dialog, depicted in Figure 2-4 part F, is constructed very similarly. It is used to export an image of the actual Dendrogram view, and hence has an additional column of settings on the left used to configure which headers are to be included, as well as Dendrogram-specific configuration in the middle column. The user can choose to include multiple or no headers at all by holding the “Apple” or “Alt” key when clicking on the list of headers. The array headers, which are ordinarily place above the array tree can be moved below it by clicking the “Below Tree?” checkbox. Using the middle column of settings, the user can indicate whether to include all genes or just the selected genes, which trees to include, whether to include the data matrix, and the X and Y pixel scaling

These interfaces provide a simple, consist way to create rich figures from the images generated by the Dendrogram display.

Description of Scatterplot display

The scatterplot display was create to solve the problem of how to compare various kinds of statistics to each other, as well as to the dendrogram clustering. Manually deriving subclusters of the data, exporting to a gene list

and graphing in other programs is a simple and effective strategy, but is slow and impractical for large numbers of statistics or clusters. Using the Generalized CDT format, the user now has the option of including the statistic as an annotation column and constructing a Scatterplot display.

Scatterplot Functionality

The scatterplot window, depicted in Figure 2-5 part A, consists of a two-dimensional scatterplot of per-gene statistics. The statistics plotted can be any of the annotation columns, any of the data columns or simply the index of the gene in the CDT file. In the plot depicted in this figure, the X axis is simply the index of the gene in the CDT file, and the Y axis is the annotation column labeled PVAL. If the data were entirely drawn from the null distribution, this plot would yield a straight line; the asympotote at low P-values indicates that a large number of genes in the data set are not behaving according to the null hypothesis. The utility of the scatterplot display is determined by the ingenuity of the researcher. For example, it can also be used to determine when spatial biases on the arrays are biasing the clustering by graphing the spot the gene was printed on against the index.

The scatterplot display supports several kinds of interaction. The currently selected genes are drawn in a different color; thus one can select genes in another view and see where they fall. Furthermore, a set of genes can be selected on the scatterplot by clicking and dragging. This causes these

same genes to be selected in other views, and also become available for export to tab-delimitted text or gene lists. Another useful feature is the ability to dynamically zoom in and out of the scatterplot using the “+” and “-“ keys. Finally, letting the mouse linger near a data point will display a tool tip with the name and coordinates of the gene.

Scatterplot Configuration

The scatterplot supports configuration through the panel directly above the plot itself, as well as through a separate “Display” dialog, which holds more detailed options. On the configuration panel, the “Order” pulldown allows the user to select the order in which the genes are draw, either selected first, selected last or row order. The “Size” pulldown selects the size of the data points. The “Dimension” checkbox allows the user to specify the desired size of the canvas that the scatterplot is drawn on, which has the effect of zooming in and out. As mentioned before, the user can achieve much the same effect using the “+” and “-“ keys. The last item on the configuration panel is the “Display…” button, which causes the “Display” dialog pictured in Figure 2-5 part B to be displayed. This same dialog is displayed if “Settings->Display…” is chosen from the menu.

The “Display” dialog has two columns which specify the extent and tick mark spacing on the X and Y axes, as well as a section dedicated to the color

settings, similar to that in the Dendrogram's "Pixel Settings" dialog pictured in Figure 2-4 part A.

Scatterplot Creation

The scatterplot must be created by the user selecting "Analysis->Make Scatterplot of Genes…". Instead of immediately creating a Scatterplot display, this action displays the the "Create Graph…" dialog at which point the user must select the desired headers to be plotted from the "X Axis" and "Y Axis" pulldown menus. The LinkedViewFrame allocates a subnode from it's ConfigNode, and actually configures it with the appropriate axis information before constructing a ScatterPanel object, binding the it to the subnode and adding it as a tab. Thus, from the beginning the ScatterPanel is associated with the columns it will draw.

Scatterplot Implementation

Similar to the Dendrogram, the Scatterplot display is implemented by the ScatterPanel container, a subclass of MainPanel, and the Views it contains. In this case, the only View is the ScatterView. There is a second component, the ScatterParameterPanel, which implements the row of configuration widgets above the scatterplot itself. Similar to the MapContainer classes, which acted as mediators between the configuration dialogs and the views in the Dendrogram display, there are AxisInfo and AxisParameter classes which are configured by the dialogs and observed by the ScatterView.

The final novel feature in the Scatterplot implementation is the SPDataSource, which specified what an object must provide in order for the ScatterView to display it. In the running application, this interface is implemented by an inner class of the ScatterPanel, and serves as a wrapper which uses the information in the View ConfigNode to extract coordinate and annotations from the DataModel. This implementation maps nicely onto the actual structure of the Xml configuration document, as consultation of parts B and C of Figure 2-1 reveal.

Scatterplot Export

Scatterplot export is fairly rudimentary. The extent of configuration required to provide professional quality charts can be daunting, and has been done many times before. The goal of the existing scatterplot export is to provide the user with a "quick and dirty" way to make a quick figure. The dialog opened by "Export->Export to Gif…" is shown in Figure 2-5 part C. The result will be more or less equivalent to a screen capture of the scatterplot. The best way to improve the scatterplot export, and possibly the functionality of scatterplot in general, would be to try and incorporate a third-party plotting package such as the Scientific Graphics Toolkit (SGT) from NOAA (http://www.epic.noaa.gov/java/sgt/).

Description of Karyoscope display

For some important types of experiments, the proximity of loci on the genome is expected to correlate with their measured values. For example, techniques such as array comparative genomic hybridization (Pollack, Perou et al. 1999) allow the detection of copy number changes on a genome scale. Displaying the data in genome ordering allows visual identification of regions of extended amplification and deletion. The Karyoscope display allows users to visualize per-gene data as a bar chart in genome order. In addition, it features configuration of gene coordinates, support for averaging loci together in various ways, and flexible display options.

Karyoscope Functionality

Two screenshots of the karyoscope are provided in Figure 2-6. Part A shows a zoomed out view and part B shows a zoomed in view around chromosome 6. There are three components to the Karyoscope display, the configuration panel, the karyoscope panel, and the familiar status panel. Moving the cursor over the karyoscope panel causes the position of the cursor, and a summary of the gene or genes contributing to the nearest locus to appear in the status panel. In the karyoscope panel itself, a crosshairs is drawn on the nearest gene and a line is drawn from the cursor to the crosshairs. This makes it very clear which gene the information is for, and makes it easy tell if it is the one you are interested in. Clicking at this point will

open a browser window directed to a web database with additional information about the gene. The final functionality offered by the karyoscope panel is the ability to zoom, either by using the “+” and “-“ keys, or by clicking and dragging a rectangular area.

Recognized Annotation

In order to display the genes in genome position, the Karyoscope display must determine the position of the loci. This is accomplished by requiring that particular annotation columns be provided. Currently, Karyoscope requires that there is a column named “CHROMOSOME”, specifying the chromosome number which must be a natural number, upon which the locus appears, a column named “ARM”, which must be one of “0”, “1”, “L” or “R”, indicating the arm that the locus appears on, and a column named “POSITION” which contains the distance from the centromere at which to render the current locus. This distance is in arbitrary units, and may be fractional. An useful future addition may be to drop the requirement for the ARM column, and to assume that the distance is from the left end of the chromosome in the absence of the ARM column.

These annotation columns need not be provided by the currently loaded Generalized CDT; as discussed later in the configuration section, they can be specified in another Generalized CDT, and be matched up using the unique identifier.

A special case is represented by yeast ORFs. If the required annotation columns are missing from the file, but the locus names conform to yeast orf name conventions, they are parsed to extract the required information, assuming that each gene occupies one map unit.

Karyoscope Configuration

The configuration of the Karyoscope display is complex, as there are several considerations in addition to the microarray data, such as the actual coordinates of the loci, the desired averaging, and what exactly to display.

The most immediate settings are provided by the components in the configuration panel. Here, the user can select the experiment to display, navigate to the previous or next experiment, set the size of the canvas, and set the number of pixels allocated to each map unit and value unit. The size of the canvas specifies exactly the pixel dimensions of the drawing surface. The pixels per map determines how long the chromosomes are in the horizontal direction. The pixels per value determines how many pixels correspond to a value of 1 in the vertical bars.

Clicking the “Display” button, or choosing “Settings->Display” from the menu will display the popup in Figure 2-7 part A. From this dialog, the user may specify what to draw for each locus, whether to use scale lines, the color settings, and how to render genes which are selected according to the shared CdtSelection object. The checkboxes on the first row specify whether to

include a connecting line between the tips of the loci, and whether to draw colored bars for each locus. The next row contains checkboxes to indicate inclusion of fold-change lines above and below the genome line. This detail makes it easy to spot loci over a particular fold change. The user must additionally specify the base of the expression data, as well as the maximum number of scale lines to draw. The third row contains color selection widgets similar to that in the Dendrogram and Scatterplot configuration, with the distinction of having more colors to configure than the other two. The “Up” and “Down” colors specify the colors in which to draw up and down colored bars, the “Genome” and “Background” colors specify the color in which to render the genome line and background respectively, and the “Line” color specifies the color in which to draw the scale lines and the connecting line.

The much simpler “Averaging” dialog, pictured in Figure 2-7 part B, can be summoned by either clicking the “Averaging…” button on the configuration panel, or by selecting “Settings->Averaging…” from the menu. There are a total of four averaging options. The first is simple no averaging at all, corresponding to the “No Averaging” button. The next, “Nearest”, averages the nearest k loci together, including the one in question. If the map coordinates so dictate, it is possible that all of these loci will be on one side of the current locus. The third option, “Neighbor”, averages the (k-1)/2 loci on the left and the (k-1)/2 loci on the right together with the current locus to produce a smoothed value. In the event that there are fewer than (k-1)/2 loci on a side, fewer loci

will be averaged together. The final option, “Interval”, allows the user to specify an interval of map units around the locus in question. All loci within this interval are averaged to calculate the value for the locus.

The final set of options, depicted in Figure 2-7 part C, can be summoned either by clicking the “Coordinates…” button of the configuration panel or by selecting “Settings->Coordinates…” off the menu. There are three ways in which coordinates can be set. Clicking on the “Load from File…” button summons a file dialog similar to that in Figure 2-2 part E. The user then may select a Generalized CDT file which will be parsed in search of the required annotation columns. Selecting “Extract from Cdt” will cause Java Treeview to attempt to extract the annotation columns from the data model itself. Finally, the available presets are listed by name. Clicking one of these buttons will cause the corresponding Generalized CDT file to be loaded, much as if it had been selected using the “Load from File…” button.

Karyoscope Creation

The Karyoscope display is created by selecting “Analysis->Make Karyoscope” from menu. In a similar sequence to that of the earlier displays, this causes allocation of a View node which is then passed into the KaryoPanel constructor. The newly created KaryoPanel is then added as a tab to the LinkedViewFrame.

Karyoscope Implementation

The Karyoscope display is provided by the KaryoPanel container, a subclass of MainPanel. The KaryoPanel container manages the shared Genome instance which tracks the locations of the loci, as well as the usual plumbing to set up and manage the views and menus which all MainPanels do. Because there are no other components to share data with, the other configuration, e.g. the averaging and what exactly to display, are stored directly in the KaryoView class.

The Genome class is the data structure which represents the positions of the loci. It can construct itself based upon an arbitrary DataModel. First, an instance of ChromsomeLocus is allocated for each row of annotations. ChromosomeLocus has slots for the original row in the CDT file, as well as the chromosome, arm and position. The original row number allows the Genome to link up the locus with the actual expression data and other annotations later on, as well as determine whether it is selected, since the CdtSelection, DataModel and the DataMatrix all work with that index. Next, the complete list of ChromosomeLocus instances are traversed to determine the number, type and size of the chromosomes. Finally, a Chromosome subclass, either LinearChromosome or CircularChromsome, is allocated and populated with loci. Thus, the Genome maintains a list of ChromosomeLocus instances in Cdt row order as well as a list of Chromosome instances, each of which contains

the loci sorted by position. This enables fast binary searches to find loci given position information, and a constant time lookup given the Cdt row number.

Some care must be taken when using a Genome object with a different DataModel than the one it was constructed from, as is done when the user specifies an alternative coordinates file. At the outset, we have a DataModel that contains the actual data and has a particular locus ordering, and a Genome constructed from a different cdt file which has its own ordering. The Genome must be updated to reflect the DataModel. First, all loci in the genome are set to the invalid index “-1”. Next, a hash from unique id to cdt row number is constructed using the DataModel. Finally, all ChromosomeLocus instances in the Genome are traversed, checking to see if the unique id is present in the hash and updating the row index as appropriate. Thus loci which do not appear in the DataModel are found to be invalid and are ignored.

It should be noted that a visualization for the CircularChromosome class has been implemented yet, although much of the mechanics, including averaging, are in place.

Karyoscope Export

Karyoscope export is initiated by selecting “Export->Export to Gif…” and is configured by the dialog show in Figure 2-7 part D. The only configuration currently supported is the selection of one or more chromosomes discontinuously from the list.

Description of Summary display

The genesis of the Summary display was a piece of user feedback. The user wanted to see the gene expression patterns along with the gene names in the results list of the gene search. Subsequently, I introduced the Summary display to show a quick view of the gene expression data of the currently selected genes.

Summary Functionality

The Summary display shows a quick summary of a subset of the data using components originally developed for the Dendrogram view. A screenshot of a typical Summary display is shown in Figure 2-8 part A. This display was produced by searching the gene annotation for the string “PHO”, selecting all, and creating a summary view. The Cdt file itself was sorted by the p-vallue from a non-parametric T-test. The p-value itself appears in the gene annotation, after the gene name. As is indicated in the figure, this display is composed of the familiar ZoomView and TextView components. All of the functionality from the Dendrogram display is carried over; thus, letting the cursor linger over the zoom view will show a tool tip containing the expression value, and clicking a gene name will open a browser window with more information about the gene.

The main difference between the Summary display and the others is that is a very lightweight display. It is show in a modeless dialog separate from the main application, and has no menu items, no configuration, and no export.

Summary Creation

The Summary display is not an independent display. It can only be created when some genes are selected, and even then only when a Dendrogram display is active. This is because the Summary display currently steals its configuration information from the Dendrogram display which was selected when it was created.

There are two ways in which the user can create a Summary display. Selecting “Analysis->Make Summary…” will summon the dialog shown in Figure Figure 2-8 part B. At this point, the user can choose to either make a summary of the currently selected genes, or to paste in a list of unique ids. In the future, it might be worthwhile to allow the user to select an arbitrary annotation column to match on. The second mechanism for creating a summary view is by pressing the “Make Popup” button on the “Search Gene Text for Substring” dialog pictured in Figure 2-2 part G. This simply makes a summary of the currently selected genes, or selecting all matching genes and then making a summary if none are selected.

Summary Implementation

The entire source code for the class which implements the Summary display, SummaryPanel, is only 126 lines of code. This is partially because of component reuse, and partially because some of the work to implement the Summary display involved changing other classes. When the construction of a new SummaryPanel is initiated, the DendroView class first determines the indexes of all genes which are to be included in the summary, and populates an integer array with them. The integer array is called the gene order array, in that the first element of the array specifies the index into the DataMatrix for the data in the first row of the summary, the second element specifies the index for the second row, and so on. This allows arbitrary subsetting and reordering of the elements in the DataMatrix, but in the context of the Summary display, it is only used to make a subset; the ordering will be the same as in the original file.

This ordering, as well as the existing ArrayDrawer, HeaderInfo and DataMatrix are used to construct and configure the SummaryPanel, which is then added to a dialog and displayed.

Concluding Remarks

In this thesis I have established a correspondence between batch and chemostat cultures at the gene expression level. I have also contributed new datasets which describe the response of batch cultures to limitation for several

nutrients. Some of these responses are predicted from the literature, and some are novel. There are further questions of interest regarding gene expression studies in the chemostat; although there is no stress response in a steady state chemostat, what effect would varying the conditions have? For instance, can we use a chemostat to unfold a temperature sensitive protein without incurring the stress response? What effect does changing a drug concentration slowly instead of quickly have?

I have also described Java Treeview, a new microarray data visualization tool which greatly assists in the interpretation of genome-scale data. These innovations set the stage for more carefully controlled gene expression studies in yeast, with the prospect of more comprehensive studies of responses common to all eukaryotes.

Code Listing 2-1: ConfigNode.java

```
jEdit - /Users/alok/Desktop/java/LinkedView/src/edu/stanford/genetics/treeview/ConfigNode.java
1  /* BEGIN_HEADER                                              Java TreeView
2   *
3   * $Author: remote $
4   * $RCSfile: ConfigNode.java,v $
5   * $Revision: 1.3 $
6   * $Date: 2003/06/23 08:03:07 $
7   * $Name: $
8   *
9   * This file is part of Java TreeView
10  * Copyright (C) 2001-2003 Alok Saldanha, All Rights Reserved.
11  *
12  * This software is provided under the GNU GPL Version 2. In particular,
13  *
14  * 1) If you modify a source file, make a comment in it containing your name and the date.
15  * 2) If you distribute a modified version, you must do it under the GPL 2.
16  * 3) Developers are encouraged but not required to notify the Java TreeView maintainers at
    alok@genome.stanford.edu when they make a useful addition. It would be nice if significant
    contributions could be merged into the main distribution.
17  *
18  * A full copy of the license can be found in gpl.txt or online at
19  * http://www.gnu.org/licenses/gpl.txt
20  *
21  * END_HEADER
22  */
23 package edu.stanford.genetics.treeview;
24
25
26 /**
27  * Defines an interface for storage of key-value pairs. Essentially all the configuration information
    for Java TreeView is stored using this interface. You will encounter two implementing classes. The
    first, most common one is an inner class of XmlConfig, which simply presents an interface to edit an
    xml document. Thus, when you mess with that inner class through this interface, you're actually
    writing XML. The second is the DummyConfigNode, which you can use for prototyping stuff or if you just
    want to use this interface to store key-value pairs in a non-persistant fashion.
28  *
29  * The easiest way to make an object persistant across different runs of the program is to bind it to a
    ConfigNode returned by XmlConfig (which is bound to a file on disk), and then just store all state
    information in the ConfigNode. Whenever the XmlConfig is saved, it will automatically save the state of
    your object. just make sure you save it before you exit!
30  *
31  * @author     Alok Saldanha <alok@genome.stanford.edu >
32  * @version $Revision: 1.3 $ $Date: 2003/06/23 08:03:07 $
33  */
34 public abstract interface ConfigNode {
35   /**
36    * create and return a subnode which has the indicated name
37    *
38    * @param  name  name for subnode
39    * @return       newly created subnode
40    */
41   public abstract ConfigNode create(String name);
42
43
44   /**
45    * fetch all nodes with the name
46    *
47    * @param  name  type of nodes to search for

                                        8/21/03 11:05 AM :: page 1
```

Code Listing 2-1 continued

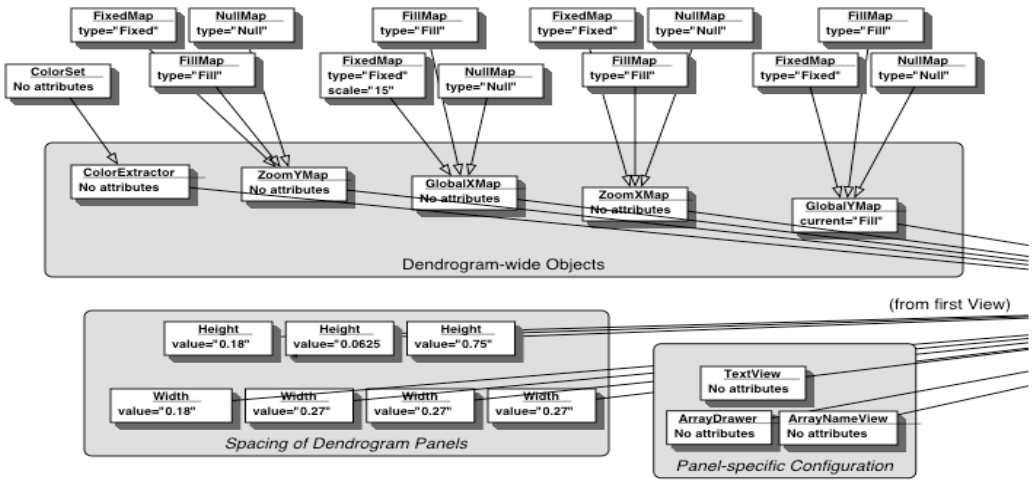
```
jEdit - /Users/alok/Desktop/java/LinkedView/src/edu/stanford/genetics/treeview/ConfigNode.java
48      * @return       array of matching nodes
49      */
50     public abstract ConfigNode[] fetch(String name);
51
52
53     /**
54      *  fetch first node by name
55      *
56      * @param  string  type of node to search for
57      * @return         first matching node
58      */
59     public abstract ConfigNode fetchFirst(String string);
60
61
62     /**
63      *  remove particular subnode
64      *
65      * @param  configNode  node to remove
66      */
67     public abstract void remove(ConfigNode configNode);
68
69
70     /**
71      *  remove all subnodes with a given name
72      *
73      * @param  string name of nodes to remove
74      */
75     public abstract void removeAll(String string);
76
77
78     /**
79      *  set attribute to be last in list
80      *
81      * @param  configNode  configNode to be made last of children
82      */
83     public abstract void setLast(ConfigNode configNode);
84
85     /**
86      * determine if a particular attribute is defined for this node.
87      *
88      * @param  string  name of attribute
89      */
90      public boolean hasAttribute(String string);
91
92
93     /**
94      *  get a double attribute
95      *
96      * @param  string  name of attribute
97      * @param  d       a default value to return
98      * @return         The attribute value
99      */
100    public abstract double getAttribute(String string, double d);
101
102
103    /**
104     *  get an int attribute

                                        8/21/03 11:05 AM :: page 2
```

Code Listing 2-1 continued

```
jEdit - /Users/alok/Desktop/java/LinkedView/src/edu/stanford/genetics/treeview/ConfigNode.java
105     *
106     * @param   string   name of attribue
107     * @param   i        default int value
108     * @return           The attribute value
109     */
110    public abstract int getAttribute (String string, int i);
111
112
113    /**
114     *  get a String attribute
115     *
116     * @param   string1   attribute to get
117     * @param   string2   Default value
118     * @return            The attribute value
119     */
120    public abstract String getAttribute (String string1, String string2);
121
122
123    /**
124     *  set a double attribute
125     *
126     * @param   att    name of attribute
127     * @param   val    The new attribute value
128     * @param   dval   The default value
129     */
130    public abstract void setAttribute (String att, double val, double dval);
131
132
133    /**
134     *  set an int attribute
135     *
136     * @param   att    name of attribute
137     * @param   val    The new attribute value
138     * @param   dval   The default value
139     */
140    public abstract void setAttribute (String att, int val, int dval);
141
142
143    /**
144     *  set a String attribute
145     *
146     * @param   att    name of attribute
147     * @param   val    The new attribute value
148     * @param   dval   The default value
149     */
150    public abstract void setAttribute (String att, String val, String dval);
151 }
152
                                8/21/03 11:05 AM :: page 3
```

Figure 2-1: Example XML Configuration Tree

A



B

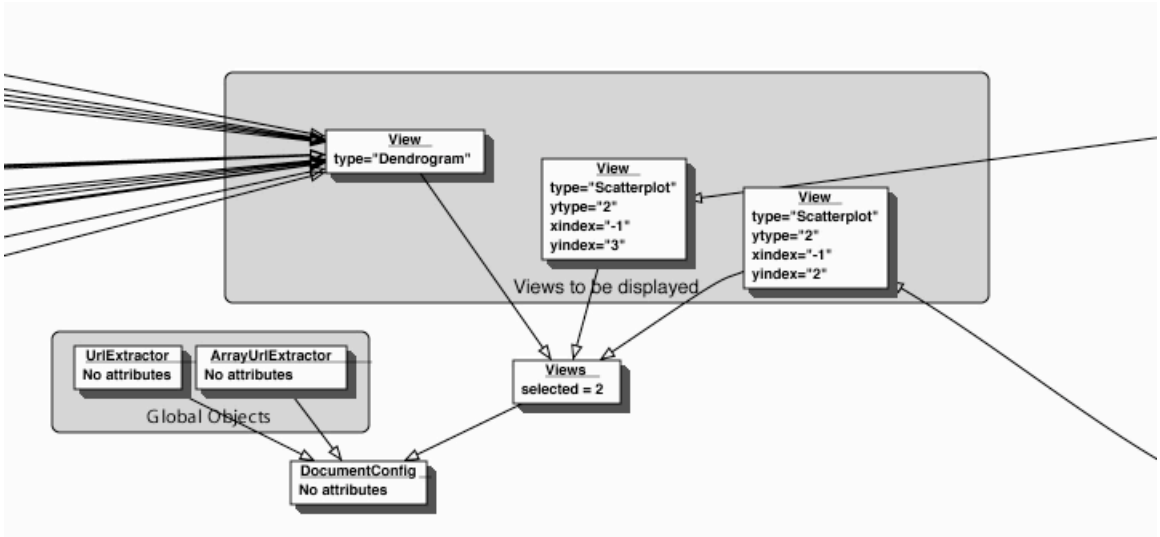


Figure 2-1 continued
C

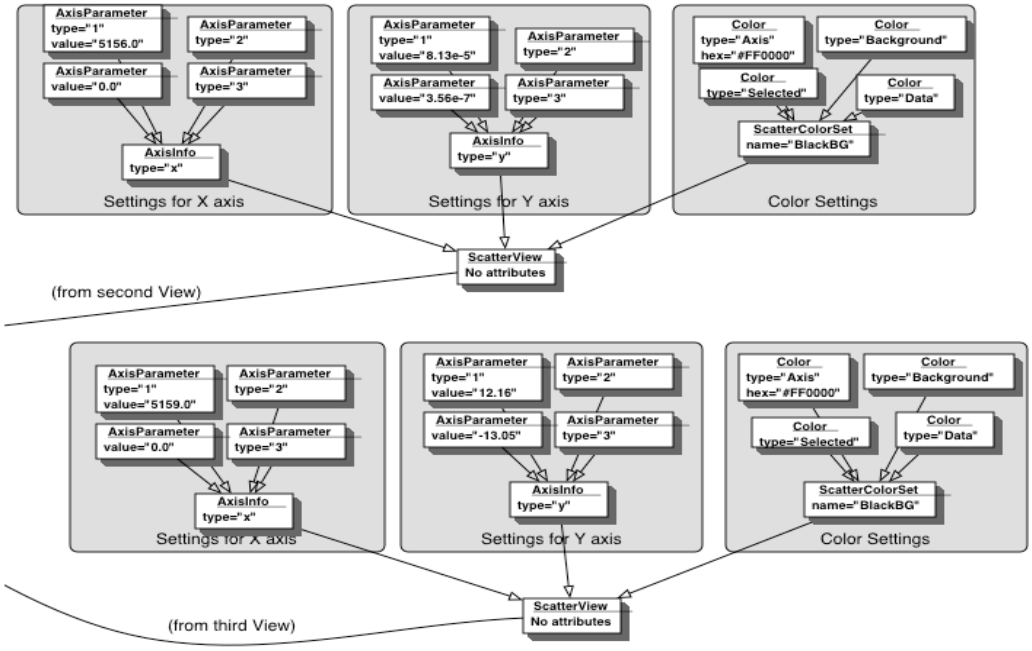


Figure 2-1. This is a graphical depiction of the contents of a typical Xml configuration file holding per-document settings. The root node is labeled “DocumentConfig” and can be found on the second page. It is the only node with no parent. When Java Treeview reads in this file, it simply parses it into an Xml tree. The root node is requested by LinkedView, which uses the Views node to determine which views to display. It then constructs the views, and binds them to the View nodes. This continues recursively. Thus, a single store() call can save the entire configuration, even though there is no centralized configuration object.

Code Listing 2-2: XmlConfig.jtv

```
                    jEdit - /Users/alok/Desktop/Alok/Research/TVFigures/XmlConfig.jtv
1  <DocumentConfig >
2  <UrlExtractor/ > <ArrayUrlExtractor/ >
3
4  <Views selected="2">
5   <View type="Dendrogram" >
6    <ColorExtractor ><ColorSet/ ></ColorExtractor >
7    <ArrayDrawer/ > <TextView/ > <ArrayNameView/ >
8    <GlobalXMap > <FixedMap type="Fixed" scale="15.0" /> <FillMap type="Fill" /> <NullMap type="Null" />
     </GlobalXMap >
9    <GlobalYMap current="Fill" > <FixedMap type="Fixed" /> <FillMap type="Fill" /> <NullMap type="Null" />
     </GlobalYMap >
10   <ZoomXMap > <FixedMap type="Fixed" /> <FillMap type="Fill" /> <NullMap type="Null" /> </ZoomXMap >
11   <ZoomYMap > <FixedMap type="Fixed" /><FillMap type="Fill" /> <NullMap type="Null" /> </ZoomYMap >
12   <Height value="0.1875" /> <Height value="0.0625" /> <Height value="0.75" />
13   <Width value="0.1818181872367859" /> <Width value="0.27272728085517883" /> <Width
     value="0.27272728085517883" /> <Width value="0.27272728085517883" />
14  </View>
15
16  <View type="Scatterplot" ytype="2" xindex="-1" yindex="2"> <ScatterView >
17   <ScatterColorSet name="BlackBG" >
18    <Color type="Background" /> <Color type="Axis" hex="#FF0000" /> <Color type="Data" /> <Color
      type="Selected" />
19   </ScatterColorSet >
20   <AxisInfo type="x">
21    <AxisParameter value="0.0" /> <AxisParameter type="1" value="5159.0" /> <AxisParameter type="2" />
      <AxisParameter type="3" />
22   </AxisInfo >
23   <AxisInfo type="y">
24    <AxisParameter value="-13.0532305160234" /> <AxisParameter type="1" value="12.1629664719857" />
      <AxisParameter type="2" /> <AxisParameter type="3" />
25   </AxisInfo >
26  </ScatterView > </View>
27
28  <View type="Scatterplot" ytype="2" xindex="-1" yindex="3"> <ScatterView >
29   <ScatterColorSet name="BlackBG" >
30    <Color type="Background" /> <Color type="Axis" hex="#FF0000" /> <Color type="Data" /> <Color
      type="Selected" />
31   </ScatterColorSet >
32   <AxisInfo type="x">
33    <AxisParameter value="0.0" /> <AxisParameter type="1" value="5156.0" /> <AxisParameter type="2" />
      <AxisParameter type="3" />
34   </AxisInfo >
35   <AxisInfo type="y">
36    <AxisParameter value="3.567608996105O4E-7" /> <AxisParameter type="1" value="8.12807101775444E-5" />
      <AxisParameter type="2" /> <AxisParameter type="3" />
37   </AxisInfo >
38  </ScatterView > </View>
39 </Views >
40 </DocumentConfig >
41
                                    8/21/03 4:07 PM :: page 1
```

Code Listing 2-2. Code listing corresponding to Figure 2-1.

Code Listing 2-3: DataMatrix.java

jEdit - /Users/alok/Desktop/java/LinkedView/src/edu/stanford/genetics/treeview/DataMatrix.java

```
1 package edu.stanford.genetics.treeview;
2
3 public interface DataMatrix {
4   double getValue(int row, int col);
5   int getNumRow();
6   int getNumCol();
7 }
```

8/21/03 7:38 PM :: page 1

Code Listing 2-3. DataMatrix interface, which provides access to the matrix of gene expression data.

Code Listing 2-4: HeaderInfo.java

jEdit - /Users/alok/Desktop/java/LinkedView/src/edu/stanford/genetics/treeview/HeaderInfo.java

```
1  package edu.stanford.genetics.treeview;
2
3
4  /**
5   * Interface to access header info about genes or arrays.
6   *
7   * @author      Alok Saldanha <alok@genome.stanford.edu >
8   * @version $Revision: 1.3 $ $Date: 2003/08/01 19:09:23 $
9   */
10 public interface HeaderInfo {
11   /**
12    *  Gets the header info for gene/array i
13    *
14    * @param   i  index of the header to get
15    * @return     The array of header values
16    */
17   public String[] getHeader(int i);
18
19   /**
20    *  Gets the header info for gene/array i, col name
21    *
22    * @param   i  index of the header to get
23    * @return     The array of header values
24    */
25   public String getHeader(int i, String name);
26
27
28   /**
29    *  Gets the names of the headers
30    *
31    * @return     The list of names
32    */
33   public String[] getNames();
34
35   /**
36   * The number of headers per gene.
37   */
38   public int getNumNames();
39   /**
40    * Gets the number of sets of headers.  This will generally be the number things which have headers,
      i.e. number of genes or number of arrays.
41    */
42   public int getNumHeaders();
43
44   /**
45    *  Gets the index associated with a particular name.
46    * Note that some header info classes may have special ways of mapping
47    * names to indexes, so that the getNames() array at the returned index
48    * may not actually match the name argument.  This is particularly true for
49    * fields like YORF, which may also be UID, etc...
50    *
51    * @param   name  A name to find the index of
52    * @return        The index value
53    */
54   public int getIndex(String name);
55 }
56
```

8/21/03 7:37 PM :: page 1

Code Listing 2-4. Code Listing of HeaderInfo interface, which provides access to the information in gene and array headers.

Code Listing 2-5: DataModel.java

```
jEdit - /Users/alok/Desktop/java/LinkedView/src/edu/stanford/genetics/treeview/DataModel.java
1  package edu.stanford.genetics.treeview;
2
3  import java.awt.MenuItem;
4  /**
5   *  This file defines the bare bones of what needs to implemented by a data model
6   *  which wants to be used with a ViewFrame and some ModelViews.
7   *
8   * @author      Alok Saldanha <alok@genome.stanford.edu >
9   * @version     @version $Revision: 1.6 $ $Date: 2003/08/01 19:09:23 $
10  */
11
12 public interface DataModel {
13     public final static double NODATA = -10000000 ;
14   public final static double EMPTY = -20000000 ;
15
16   /**
17    *  Gets the documentConfig attribute of the DataModel object
18    *
19    * values stored in the <code>ConfigNode </code>s of this <code>XmlConfig </code> should be persistent
      across multiple openings of this DataModel.
20    * @return     The documentConfig value
21    */
22   public XmlConfig getDocumentConfig ();
23
24
25   /**
26    *  Gets the file path or url which this <code>DataModel </code> was built from.
27    *
28    * @return     String representation of file path or url
29    */
30   public String getSource ();
31
32
33   /**
34    *  Gets the fileSet which this <code>DataModel </code> was built from.
35    *
36    * @return     The actual <code>Fileset </code> which generated this <code>DataModel </code>
37    */
38   public FileSet getFileSet ();
39
40
41   /**
42    *  Gets the HeaderInfo associated with genes for this DataModel.
43    *
44     * There are two special indexes, YORF and NAME, which mean the unique id column and the description
      column, respectively. See TVModel.TVModelHeaderInfo for details.
45     */
46   public HeaderInfo getGeneHeaderInfo ();
47
48
49   /**
50    *  Gets the HeaderInfo associated with arrays for this DataModel.
51    */
52   public HeaderInfo getArrayHeaderInfo ();
53
54
55   /**

                                    8/21/03 7:39 PM :: page 1
```

Code Listing 2-5 continued

```
      jEdit - /Users/alok/Desktop/java/LinkedView/src/edu/stanford/genetics/treeview/DataModel.java
56     *  This not-so-object-oriented  hack is in those rare instances where it is not
57     *  enough to know that we've got a DataModel.
58     *
59     * @return     a string representation  of the type of this <code>DataModel </code>
60     */
61    public String getType();
62
63
64    /**
65     *  Gets a menu item which pops up a window with some stats for this <code>DataModel </code> object
66     *
67     * @return     A MenuItem, complete with listener.
68     */
69    public MenuItem getStatMenuItem();
70
71    /**
72    * returns the datamatrix which underlies this data model,
73    * typically the matrix of measured intensity ratios.
74    */
75    public DataMatrix getDataMatrix();
76 }
77
                                    8/21/03 7:39 PM :: page 2
```

Code Listing 2-5. DataModel interface, which describes the methods which are required for other classes to create views of the data.

Code Listing 2-6: MainPanel.java

jEdit - /Users/alok/Desktop/java/LinkedView/src/edu/stanford/genetics/treeview/MainPanel.java

```
1  package edu.stanford.genetics.treeview;
2  import java.awt.Menu;
3
4  /**
5   * implementing objects are expected to be subclasses of component. The purpose
6   * of this class is to provide an interface for LinkedView, whereby different views
7   * can be added to a tabbed panel. This will be useful for other applications where
8   * you want to manipulate generic view.
9   *
10  * @author      Alok Saldanha <alok@genome.stanford.edu >
11  * @version $Revision: 1.4 $ $Date: 2003/06/22 08:39:03 $
12  */
13 public interface MainPanel {
14   /**
15    * This syncronizes the sub compnents with their persistent storage.
16    */
17   public void syncConfig ();
18
19   /**
20   * this method gets the config node on which this component is based, or null.
21   */
22   public ConfigNode getConfigNode ();
23
24   /**
25    * Add items related to settings
26    *
27    * @param  menu  A menu to add items to.
28    */
29   public void populateSettingsMenu (Menu menu);
30
31
32   /**
33    * Add items which do some kind of analysis
34    *
35    * @param  menu  A menu to add items to.
36    */
37   public void populateAnalysisMenu (Menu menu);
38
39
40   /**
41    * Add items which allow for export, if any.
42    *
43    * @param  menu  A menu to add items to.
44    */
45   public void populateExportMenu (Menu menu);
46
47
48   /**
49    * ensure a particular index is visible. Used by Find.
50    *
51    * @param  i  Index of gene in cdt to make visible
52    */
53   public void scrollToIndex (int i);
54 }
55
```

8/28/03 2:21 PM :: page 1

Code Listing 2-7: ModelView.java

```
      jEdit - /Users/alok/Desktop/java/LinkedView/src/edu/stanford/genetics/treeview/ModelView.java
1
2   package edu.stanford.genetics.treeview;
3
4
5   import javax.swing.*;
6   import java.awt.*;
7   import java.awt.event.*;
8   import java.awt.image.*;
9
10  /**
11  * superclass, to hold info and code common to all model views
12  *
13  * This currently consists of buffer management, status and hints panels.
14  * Interestingly, but necessarily, it has no dependancy on any models.
15  */
16  public abstract class ModelView extends JPanel implements java.util.Observer,
17  MouseListener {
18    protected ViewFrame      viewFrame = null;
19    protected MessagePanel   hint = null;
20    protected MessagePanel   status = null;
21    protected boolean        hasMouse = false;
22
23    /* here so that subclass will work with BufferedModelView too */
24    protected boolean        offscreenValid     = false;
25    protected boolean        offscreenChanged   = false;
26    protected Dimension      offscreenSize      = null;
27    /**
28    * holds actual thing to be displayed...
29    */
30    protected JComponent  panel;
31
32    protected ModelView () {
33      super(false);
34      setBackground (Color.white);
35    }
36
37    /**
38    * viewName, returns name of view suitable for printing
39    * perhaps this should be replaced by reflection?
40    *
41    * @return String containing name of view.
42    */
43    abstract public String viewName();
44
45    public void setViewFrame (ViewFrame m) {viewFrame = m;}
46    public ViewFrame getViewFrame () {return viewFrame; }
47    public void setHintPanel (MessagePanel h) {hint = h;}
48    public void setStatusPanel (MessagePanel s) {status = s;}
49
50    private String[] default_hint = null;
51    public String[]  getHints () {
52      if (default_hint == null) {
53        default_hint = new String [] {"No hints for " + viewName()};
54      }
55      return default_hint;
56    }
57

                                    8/21/03 8:47 PM :: page 1
```

Code Listing 2-7 continued

```
jEdit - /Users/alok/Desktop/java/LinkedView/src/edu/stanford/genetics/treeview/ModelView.java
58    private String[] default_status = null;
59    /**
60    * Strings describing status to user, suitable for display.
61    *
62    * @return Array of strings, representing status
63    */
64    public String[] getStatus () {
65      if (default_status == null) {
66        default_status = new String []
67        {"No status info for " + viewName ()};
68      }
69      return default_status;
70    }
71
72
73    public JComponent getComponent () {return panel;}
74
75
76
77    /**
78    * Update the double buffer, if buffered
79    * Otherwise, just called by paintComponent to paint the main component.
80    *
81    * called only when offscreen buffer is marked as invalid, or if
82    * the onscreen size has changed.
83    *
84    *
85    * note: now actually called by paintcomponent to update the swing double buffer.
86
87    */
88    abstract protected void updateBuffer (Graphics g);
89
90    /**
91    * This is a stub so that components which work with this will also work with the ModelViewBuffered.
92    * importantly, no buffer is ever actually allocated.
93    */
94    public synchronized void paintComponent (Graphics g) {
95      Rectangle clip = g.getClipRect ();
96      g.setColor (Color.white);
97      g.fillRect (clip.x,clip.y,clip.width,   clip.height);
98
99
100     Dimension reqSize = getSize ();
101     if (reqSize == null) { return;}
102     // monitor size changes
103     if ((offscreenSize == null) ||
104       (reqSize.width != offscreenSize.width) ||
105       (reqSize.height != offscreenSize.height)) {
106         offscreenChanged = true;
107         offscreenSize = reqSize;
108       }
109
110       if (isEnabled ()) {
111         offscreenValid = false;
112         updateBuffer (g);
113         paintComposite (g);
114       }

                                    8/21/03 8:47 PM :: page 2
```

Code Listing 2-7 continued

```
jEdit - /Users/alok/Desktop/java/LinkedView/src/edu/stanford/genetics/treeview/ModelView.java
115         //  System.out.println("Exiting   " + viewName()  + " to clip  " + clip );
116     }
117
118     /**
119     This call is to be used to add a quick addition to the
120     component which you don't want to put on the doublebuffer.  The
121     composite could potentially be another buffer.
122
123     Currently, this is only used by globalview for adding the zoom
124     rect and focus rect.
125     */
126     public void paintComposite (Graphics g) {
127       return;
128     }
129
130     public void addNotify () {
131       super.addNotify ();
132     }
133
134     /**
135     * This does the following:
136     *  1) requests focus
137     *  2) sets status and hint panels appropriately
138     *  3) keeps track of whether we have the mouse.
139     */
140     public void mouseEntered (MouseEvent e) {
141       if (viewFrame.windowActive ()) {
142         requestFocus ();
143         if (hint != null) {hint.setMessages (getHints ());}
144         try {
145           if (status != null) {status.setMessages (getStatus ());}
146         } catch (Exception ex) {
147           JOptionPane.showMessageDialog (this, ex.toString ());
148         }
149       }
150       hasMouse = true;
151     }
152     /**
153     * keeps track of when mouse not present.
154     */
155     public void mouseExited (MouseEvent e) {
156       hasMouse = false;
157     }
158     /* a bunch of stubs so we can claim to be a MouseListener */
159     public void mouseClicked (MouseEvent e) {}
160     public void mousePressed (MouseEvent e) {}
161     public void mouseReleased (MouseEvent e) {}
162     public void mouseMoved (MouseEvent e) {}
163     public void mouseDragged (MouseEvent e) {}
164     public void keyReleased (KeyEvent e) {}
165     public void keyTyped (KeyEvent e) {}
166 }
167
                                    8/21/03 8:47 PM :: page 3
```

Code Listing 2-7. ModelView is an abstract class which describes the methods which must be implemented in order for a MainPanel to effectively use a View. It also provides a lot of useful functionality, and Views which use it can easily

acquire persistent offscreen buffering by implementing a subclass, BufferedModelView.

Code Listing 2-8: ViewFrame.java

```
jEdit - /Users/alok/Desktop/java/LinkedView/src/edu/stanford/genetics/treeview/ViewFrame.java
1  package edu.stanford.genetics.treeview;
2
3  import java.awt.*;
4  import java.awt.event.*;
5  import java.io.*;
6  import java.net.*;
7  import java.util.*;
8  import javax.swing.*;
9
10 import edu.stanford.genetics.treeview.dendroview.ColorPresets;
11
12 /**
13  *  Any frame that wants to contain MainPanels must extend this.
14  *
15  * @author     Alok Saldanha <alok@genome.stanford.edu >
16  * @version    @version $Revision: 1.11 $ $Date: 2003/08/22 16:38:32 $
17  */
18 public abstract class ViewFrame extends JFrame implements Observer {
19   // must override in subclass...
20   /**
21    *  This is to ensure that we can observe the MainPanels when they change.
22    *
23    * @param  observable  The MainPanel or other thing which changed.
24    * @param  object      Generally null.
25    */
26   public abstract void update(Observable observable, Object object);
27
28
29   /**
30    *  Sets up a <code>FileMru</code> using a particular config node.
31    * This <code>FileMru</code> can later be edited or used to show a mru menu.
32    *
33    * @param  fileMruNode  Node which will be bound to the FileMru
34    */
35   protected void setupFileMru (ConfigNode fileMruNode) {
36     fileMru = new FileMru();
37     fileMru.bindConfig (fileMruNode);
38     try {
39       fileMru.removeMoved ();
40     } catch (Exception e) {
41       LogPanel.println ("problem checking MRU in ViewFrame constructor: " + e.toString());
42       e.printStackTrace ();
43     }
44
45     fileMru.addObserver (this);
46     fileMru.notifyObservers ();//sends us message
47   }
48
49
50   /**
51    *  Centers the frame onscreen.
52    *
53    * @param  rectangle  A rectangle describing the outlines of the screen.
54    */
55   private void center(Rectangle rectangle) {
56     Dimension dimension  = getSize();

8/22/03 12:17 PM :: page 1
```

Code Listing 2-8 continued

```
jEdit - /Users/alok/Desktop/java/LinkedView/src/edu/stanford/genetics/treeview/ViewFrame.java
57      setLocation ((rectangle.width  - dimension.width)  / 3 + rectangle.x,  (rectangle.height  -
        dimension.height)  / 3 + rectangle.y);
58    }
59
60
61    /**  Determines  dimension  of  screen  and  centers  frame  onscreen.  */
62    public void centerOnscreen () {
63      // trying this for mac...
64      Toolkit  toolkit       = Toolkit. getDefaultToolkit ();
65      Dimension  dimension   = toolkit. getScreenSize ();
66      Rectangle  rectangle   = new Rectangle (dimension);
67
68      // XXX should drag out of global config
69      setSize (rectangle.width  * 3 / 4, rectangle.height  * 4 / 5);
70      center (rectangle);
71    }
72
73
74    /**  Sets a listener  on self,  so taht we can grab  focus  when  activated,  and close  ourselves  when
      closed. */
75    private  void setupWindowListener  () {
76      addWindowListener  (
77        new WindowAdapter () {
78          public  void windowActivated (WindowEvent  windowEvent)  {
79            setWindowActive (true);
80          }
81
82
83          public  void windowClosing (WindowEvent  windowEvent)  {
84            closeWindow ();
85          }
86
87
88          public  void windowDeactivated (WindowEvent  windowEvent)  {
89            setWindowActive (false);
90          }
91        });
92    }
93
94
95    /**
96     *  Constructor  for the ViewFrame  object
97     * Sets title and window listeners
98     *
99     * @param  title  Title  for the viewframe.
100    */
101   public ViewFrame (String  title)  {
102     super (title);
103     setupWindowListener  ();
104   }
105
106
107   /**  construts  an untitled  <code>ViewFrame </code> */
108   public ViewFrame () {
109     super ();
110     setupWindowListener  ();
111   }
                                        8/22/03 12:17 PM :: page 2
```

Code Listing 2-8 continued

```
jEdit - /Users/alok/Desktop/java/LinkedView/src/edu/stanford/genetics/treeview/ViewFrame.java
112
113
114   /**
115    *  Keep track of when active, so that clicks don't get passed through too much.
116    *
117    * @param   flag  The new windowActive value
118    */
119   protected void setWindowActive (boolean flag) {
120     windowActive = flag;
121   }
122
123
124   /**
125    *  Keep track of when active, so that clicks don't get passed through too much.
126    *
127    * @return     True if window is active.
128    */
129   public boolean windowActive () {
130     return windowActive;
131   }
132
133
134   /**  Keep track of when active, so that clicks don't get passed through too much. */
135   private boolean windowActive;
136
137
138   /**  close window cleanly.
139   * causes documentConfig to be stored.
140   */
141   public void closeWindow () {
142     try {
143       DataModel dataModel = getDataModel ();
144       if (dataModel != null) {
145         XmlConfig documentConfig = dataModel. getDocumentConfig ();
146         if (documentConfig != null) {
147           documentConfig. store ();
148         }
149       }
150     } catch (Exception e) {
151       System.out. println ("ViewFrame.closeWindow()  Got exception: " + e);
152     }
153     dispose ();
154   }
155
156
157   /**
158    *  required by all <code>ModelPanel </code>s
159    *
160    * @return   The shared CdtSelection object.
161    */
162   public CdtSelection getCdtSelection () {
163     return cdtSelection;
164   }
165
166
167   /**
168    *  used by data model to signal completion of loading.

8/22/03 12:17 PM :: page 3
```

Code Listing 2-8 continued

```
jEdit - /Users/alok/Desktop/java/LinkedView/src/edu/stanford/genetics/treeview/ViewFrame.java
169     * The <code>ViewFrame</code> will react by reconfiguring it's widgets.
170     *
171     * @param  b  The new loaded value
172     */
173    public abstract void setLoaded(boolean b);
174
175
176    /**
177     *  returns special nodata value.
178     *  generally, just cribs from the <code>DataModel</code>
179     *
180     * @return     A special double which means nodata available.
181     */
182
183    public abstract double noData();
184
185
186    /**
187     *  returns the UrlPresets for the views to make use of when configuring linking
188     *  for genes
189     *
190     * @return     The shared <code>UrlPresets</code> object for genes
191     */
192    public abstract UrlPresets getGeneUrlPresets();
193
194
195    /**
196     *  returns the UrlPresets for the views to make use of when configuring linking
197     *  for arrays
198     *
199     * @return     The shared <code>UrlPresets</code> object for arrays
200     */
201    public abstract UrlPresets getArrayUrlPresets();
202
203
204    /**
205     *  returns a ColorExtractor for the views to make use of when drawing color maps
206     *  of array values.
207     *
208     * @return     The shared <code>ColorPresets</code> object
209     */
210    public abstract ColorPresets getColorPresets();
211
212
213    /**
214     *  Gets the loaded attribute of the ViewFrame object
215     *
216     * @return     True if there is currently a model loaded.
217     */
218    public abstract boolean getLoaded();
219
220
221    /**
222     *  Gets the shared <code>DataModel</code>
223     *
224     * @return     Gets the shared <code>DataModel</code>
225     */

                                    8/22/03 12:17 PM :: page 4
```

Code Listing 2-8 continued

```
jEdit - /Users/alok/Desktop/java/LinkedView/src/edu/stanford/genetics/treeview/ViewFrame.java
226  public abstract DataModel getDataModel ();
227
228
229  /**
230   * Should scroll all MainPanels in this view frame to the specified gene. Default
231   * implementation does nothing.
232   *
233   * @param  i  gene index in model to scroll the mainpanel to.
234   */
235  public void scrollToIndex (int i) {
236  }
237
238
239  /** The shared selection object */
240  CdtSelection cdtSelection = null;
241
242
243  /**
244   * url linking support
245   *
246   * @param  i  index of gene who's url you would like to display.
247   */
248  public void displayURL (int i) {
249    displayURL (getUrl (i));
250  }
251
252
253  /**
254   * Gets the url for a particular gene.
255   *
256   * @param  i  index of the gene, for the gene's <code>UrlExtractor </code>
257   * @return    A string representation of the url
258   */
259  public String getUrl (int i) {
260    if (urlExtractor == null) {
261      return null;
262    }
263    return urlExtractor. getUrl (i);
264  }
265
266
267  /**
268   * Gets the url for a particular array.
269   *
270   * @param  i  index of the array, for the array's <code>UrlExtractor </code>
271   * @return    A string representation of the url
272   */
273  public String getArrayUrl (int i) {
274    if (arrayUrlExtractor == null) {
275      return null;
276    }
277    return arrayUrlExtractor. getUrl (i);
278  }
279
280
281  /**
282   * Pops up a browser window with the specified url

                                    8/22/03 12:17 PM :: page 5
```

Code Listing 2-8 continued

```
jEdit - /Users/alok/Desktop/java/LinkedView/src/edu/stanford/genetics/treeview/ViewFrame.java
283     *
284     * @param  string  String representation  of the url.
285     */
286   public void displayURL (String string) {
287     if (string == null) {
288       return ;
289     }
290     try {
291       URL url  = new URL(string);
292       string = url.toString ();
293       if (browserControl  == null) {
294         browserControl  = BrowserControl. getBrowserControl ();
295       }
296
297       browserControl. displayURL (string);
298     } catch (MalformedURLException  e) {
299       LogPanel. println (new StringBuffer ("Malformed  url: "). append (e). toString ());
300     } catch (IOException  e) {
301       LogPanel. println (new StringBuffer ("Could  not  load  url: "). append (e). toString ());
302     }
303
304   }
305
306
307   /**
308    *  Gets the UrlExtractor  for the arrays.
309    *
310    * This object is used to convert a given array index into a url string. It can be configured  to do
       this in multiple  ways.
311    *
312    * @return     The UrlExtractor  for the arrays
313    */
314   public UrlExtractor  getArrayUrlExtractor () {
315     return arrayUrlExtractor;
316   }
317
318
319   /**
320    *  Gets the UrlExtractor  for the genes.
321    *
322    * This object is used to convert a given gene index into a url string. It can be configured  to do
       this in multiple  ways.
323    *
324    * @return     The UrlExtractor  for the genes
325    */
326   public UrlExtractor  getUrlExtractor () {
327     return urlExtractor;
328   }
329
330
331   /**
332    *  Sets the arrayUrlExtractor  attribute  of the ViewFrame  object
333    *
334    * @param  ue  The new arrayUrlExtractor  value
335    */
336   public void setArrayUrlExtractor (UrlExtractor  ue) {
337     arrayUrlExtractor  = ue;

                                        8/22/03 12:17 PM :: page 6
```

Code Listing 2-8 continued

```
jEdit - /Users/alok/Desktop/java/LinkedView/src/edu/stanford/genetics/treeview/ViewFrame.java
338   }
339
340
341   /**
342    *  Sets the urlExtractor attribute of the ViewFrame object
343    *
344    * @param  ue  The new urlExtractor value
345    */
346   public void setUrlExtractor (UrlExtractor ue) {
347     urlExtractor = ue;
348   }
349
350   abstract public GeneFinder getGeneFinder ();
351
352 /**
353 * Open a dialog which allows the user to select a new data file
354 *
355 * @return The fileset corresponding to the dataset.
356 */
357       protected FileSet offerSelection ()
358     throws LoadException
359     {
360       FileSet fileSet1; // will be chosen...
361
362       JFileChooser fileDialog = new JFileChooser ();
363       CdtFilter ff = new CdtFilter ();
364       try {
365         fileDialog.addChoosableFileFilter (ff);
366         // will fail on pre-1.3 swings
367         fileDialog.setAcceptAllFileFilterUsed (true);
368       } catch (Exception e) {
369         // hmm... I'll just assume that there's no accept all.
370         fileDialog.addChoosableFileFilter (new javax.swing.filechooser. FileFilter () {
371           public boolean accept (File f) {
372             return true;
373           }
374           public String getDescription () {
375             return "All Files";
376           }
377         });
378       }
379       fileDialog.setFileFilter (ff);
380       fileDialog.setFileSelectionMode (JFileChooser.FILES_ONLY);
381       String string = fileMru.getMostRecentDir ();
382       if (string != null) {
383         fileDialog.setCurrentDirectory (new File(string));
384       }
385       int retVal = fileDialog.showOpenDialog (this);
386       if (retVal == JFileChooser.APPROVE_OPTION)  {
387         File chosen = fileDialog.getSelectedFile ();
388         fileSet1 = new FileSet(chosen.getName (), chosen.getParent ()+File.separator);
389       } else {
390         throw new LoadException ("File Dialog closed without selection..." , LoadException.NOFILE);
391       }
392       return fileSet1;
393     }
394
                                    8/22/03 12:17 PM :: page 7
```

Code Listing 2-8 continued

```
jEdit - /Users/alok/Desktop/java/LinkedView/src/edu/stanford/genetics/treeview/ViewFrame.java
395
396
397
398   /**
399    *  Rebuild a particular window menu.
400    *
401    * @param  windows  the list of windows to add elements to.
402    *
403    * Add a menu item for each window which grants that window the foreground when selected.
404    */
405   public void rebuildWindowMenu (Vector windows) {
406     windowMenu. removeAll ();
407     MenuItem closeItem = new MenuItem ("Close Window", new MenuShortcut (KeyEvent.VK_W));
408     closeItem. addActionListener (new ActionListener () {
409       public void actionPerformed (ActionEvent actionEvent) {
410         closeWindow ();
411       }
412     });
413     MenuItem newItem = new MenuItem ("New Window", new MenuShortcut (KeyEvent.VK_N));
414     newItem. addActionListener (new ActionListener () {
415       public void actionPerformed (ActionEvent actionEvent)
416       {
417         createNewFrame ();
418       }
419     });
420
421     int max  = windows. size();
422     for (int i = 0; i < max; i++) {
423       if (i > 8) {
424         return;
425       }// just want first 9 windows...
426       try {
427         MenuItem focusItem  = getFocusItem (windows, i);
428         windowMenu. add(focusItem);
429       } catch (Exception e) {
430         System.out. println ("TreeView.rebuildWindowMenu()  got exception " + e);
431       }
432     }
433    windowMenu. addSeparator ();
434    windowMenu. add(newItem);
435    windowMenu. add(closeItem);
436   }
437
438   /**
439   * currenlty, only the concrete subclass has a reference to the application, and hence can create new
      frames.
440   * perhaps this will change if I add an interface for the App classes.
441   */
442   public abstract void createNewFrame ();
443
444   /**
445   *  Constructs a MenuItem which causes the i'th window to be moved to the front.
446   *
447   * @param  windows  a list of windows
448   * @param  i  which window to move to the front.
449   * @return    a menuItem which focuses the i'th window, or null if more than 9 windows.
450   */

                                    8/22/03 12:17 PM :: page 8
```

Code Listing 2-8 continued

```
jEdit - /Users/alok/Desktop/java/LinkedView/src/edu/stanford/genetics/treeview/ViewFrame.java
451   private MenuItem getFocusItem(Vector windows, int i) {
452     int p1                  = i + 1;
453     if (p1 > 9) {
454       return null;
455     }
456     final ViewFrame source  = (ViewFrame) windows.elementAt(i);
457     String name;
458     if (source.getLoaded()) {
459       name = source.getDataModel().getFileSet().getRoot();
460     } else {
461       name = "Not Loaded";
462     }
463     MenuItem focusItem      = new MenuItem(name, new MenuShortcut(getKey(p1)));
464     focusItem.addActionListener(
465     new ActionListener() {
466       public void actionPerformed(ActionEvent e) {
467         source.toFront();
468       }
469     });
470     return focusItem;
471   }
472
473
474   /**
475   * Gets the key corresponding to a particular number.
476   *
477   * @param  i  The number
478   * @return    The VK_blah key value
479   */
480   protected int getKey(int i) {
481     switch (i) {
482      case 0:
483        return KeyEvent.VK_0;
484        case 1:
485        return KeyEvent.VK_1;
486      case 2:
487        return KeyEvent.VK_2;
488      case 3:
489        return KeyEvent.VK_3;
490      case 4:
491        return KeyEvent.VK_4;
492      case 5:
493        return KeyEvent.VK_5;
494      case 6:
495        return KeyEvent.VK_6;
496      case 7:
497        return KeyEvent.VK_7;
498      case 8:
499        return KeyEvent.VK_8;
500      case 9:
501        return KeyEvent.VK_9;
502     }
503     return 0;
504   }
505
506   protected Menu windowMenu;
507
                                        8/22/03 12:17 PM :: page 9
```

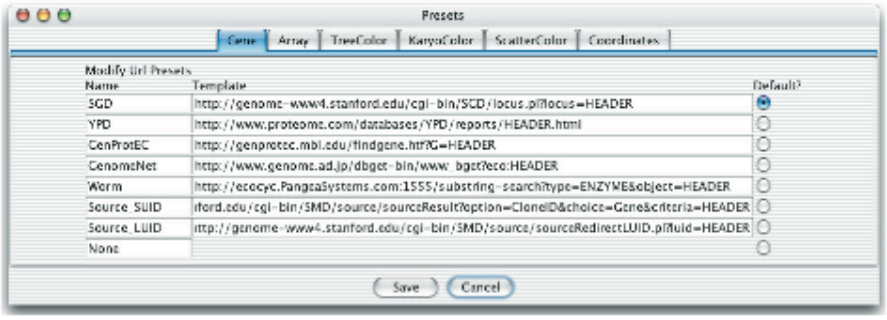
Code Listing 2-8 continued

```
jEdit - /Users/alok/Desktop/java/LinkedView/src/edu/stanford/genetics/treeview/ViewFrame.java
508
509
510     /**  The global most recently used object.  */
511     protected FileMru fileMru;
512     /**  allows opening of urls in external browser */
513     protected BrowserControl browserControl = null;
514     /**  url extractor for genes */
515     private UrlExtractor urlExtractor;
516     /**  url extractor for arrays */
517     private UrlExtractor arrayUrlExtractor;
518
519 }
520
                                    8/22/03 12:17 PM :: page 10
```

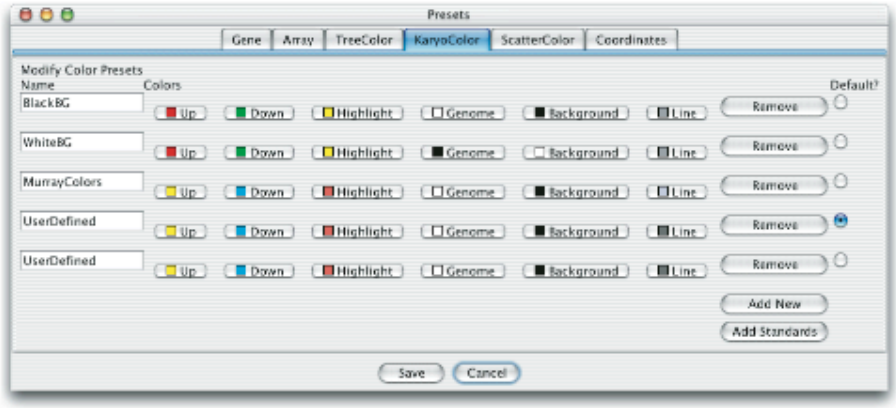
Code Listing 2-8. The ViewFrame abstract class dictates the functionality which is common to all ViewFrames, and provides some of that functionality itself.

Figure 2-2: ViewFrame Dialogs

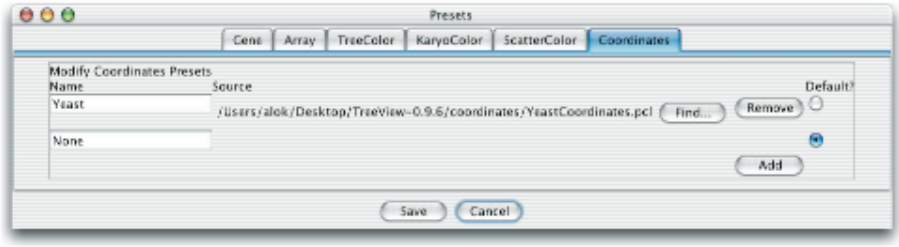
A



B



C



D

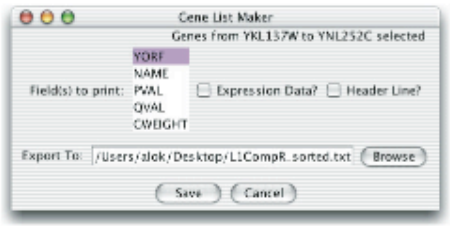


Figure 2-2 continued

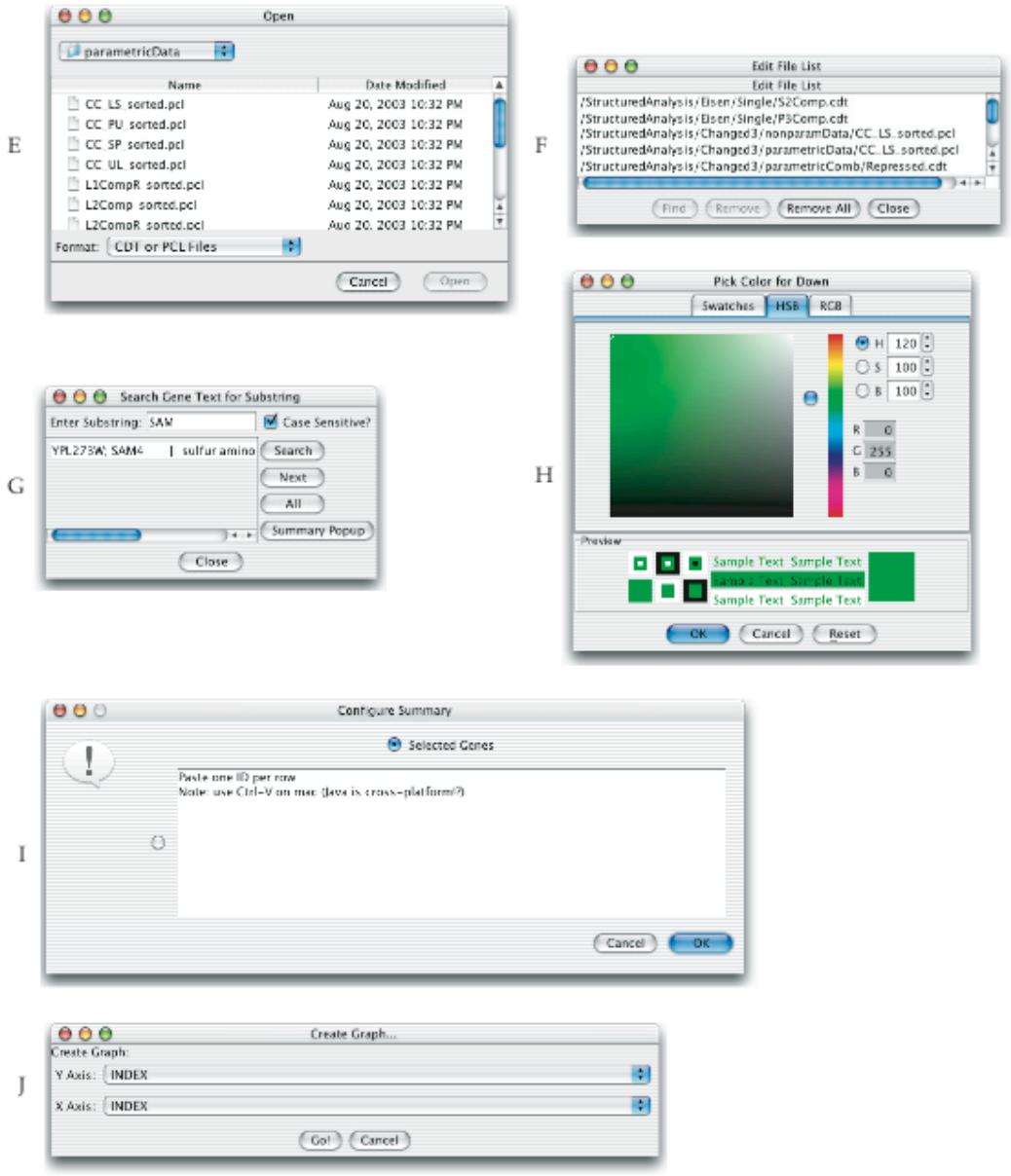


Figure 2-2. This figure depicts dialog windows which are accessible from most ViewFrame windows, and in particular the LinkedViewFrame. See the section entitled “Description of ViewFrame functionality” for details.

Figure 2-3: Dendrogram Display

Figure 2-3. This figure depicts the Dendrogram view of the Linkedview application. The container class which fills the window and manages the borders is DendroView. It delegates drawing of the different areas to other classes. A descriptive name and the java class is provided for each of the areas. Note that the same java class is used differently to provide status and usage hints.

Figure 2-4: Dendrogram Dialogs

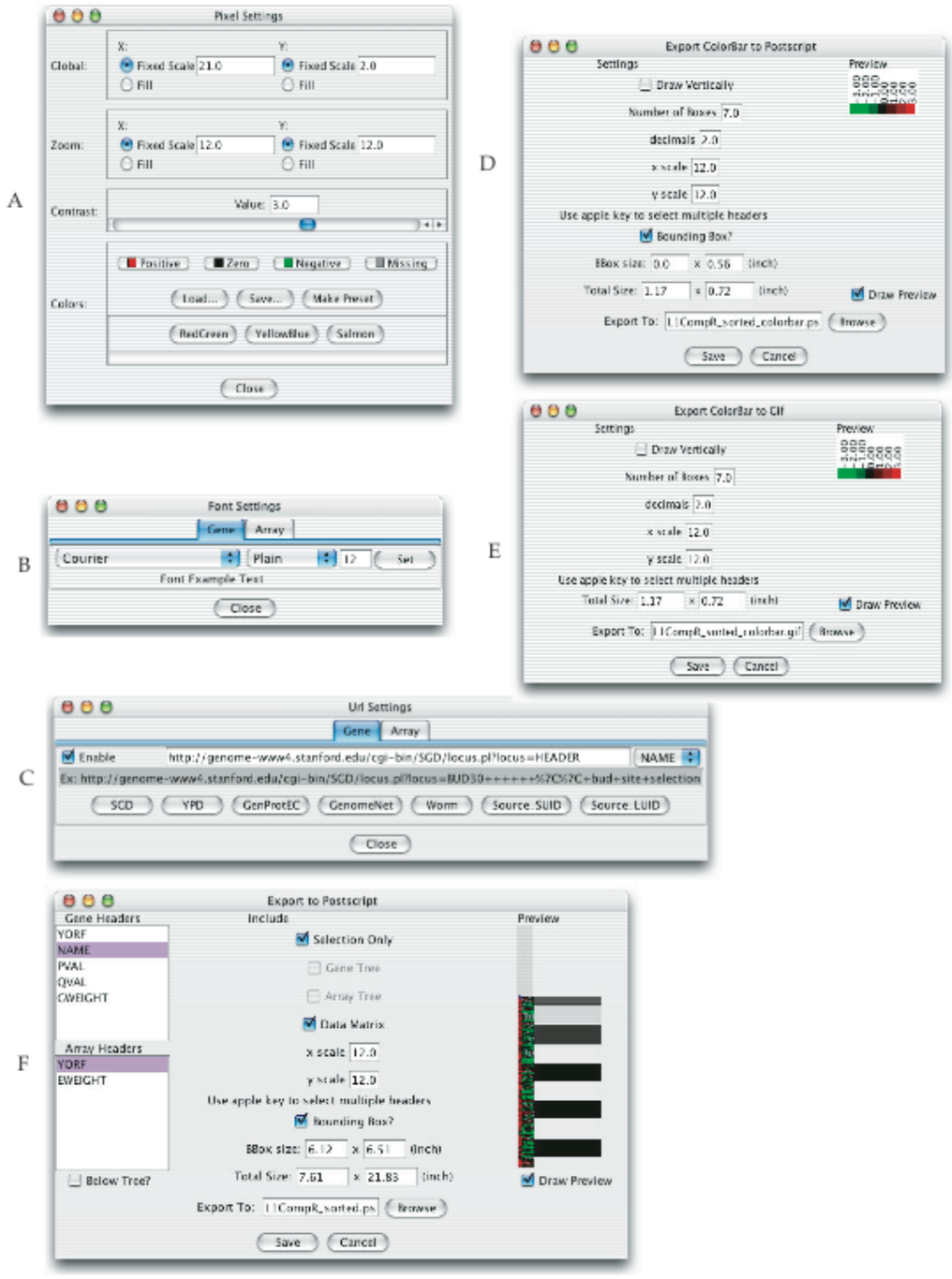


Figure 2-5: Scatterplot Display and Dialogs

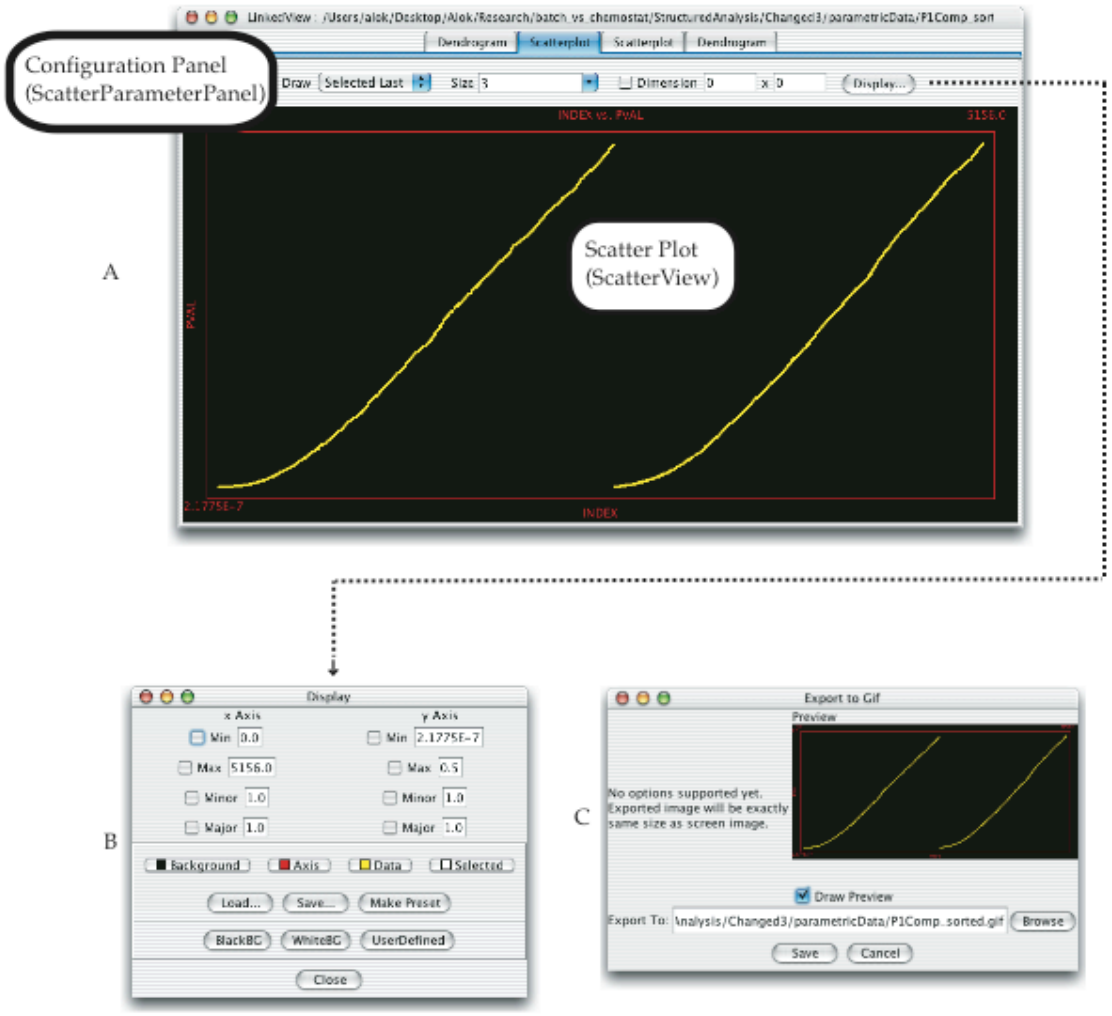
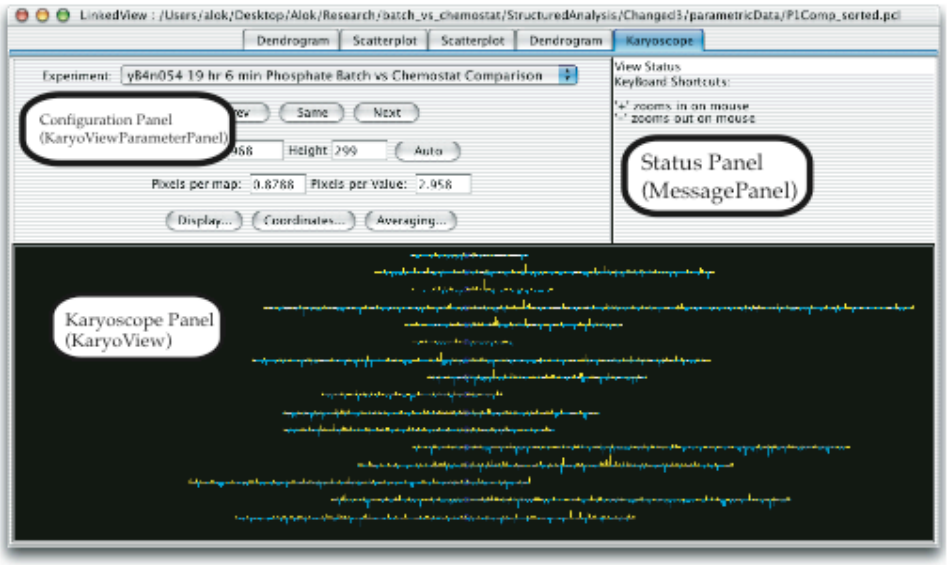


Figure 2-6: Karyoscope Display

A



B

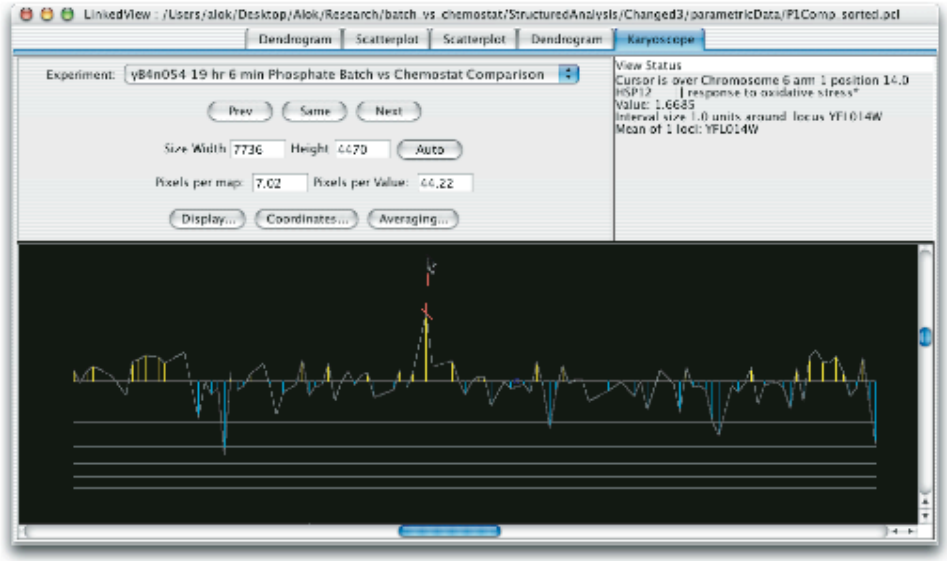


Figure 2-7: Karyoscope Dialogs

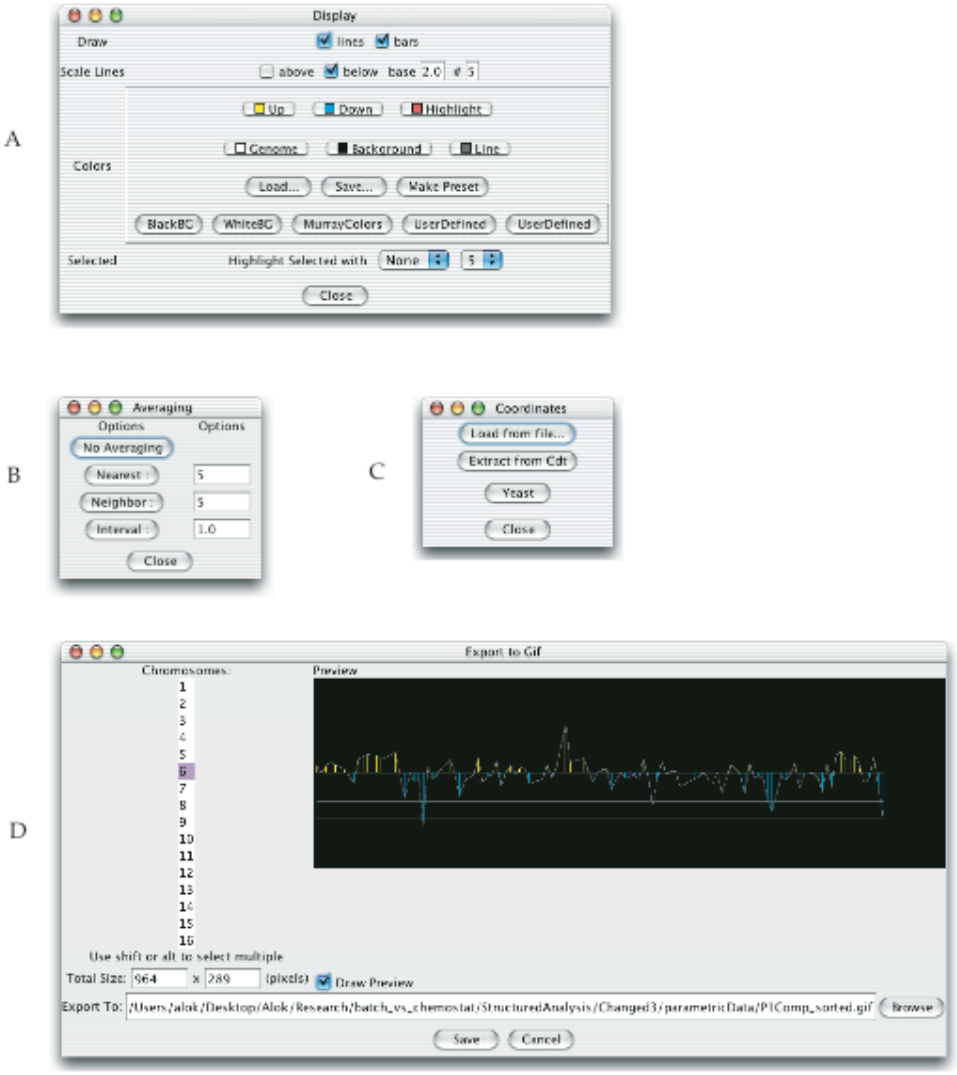
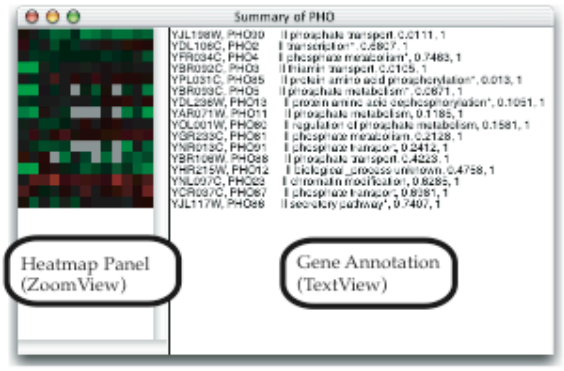
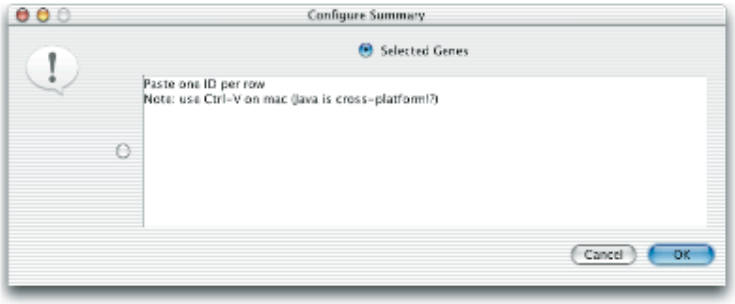


Figure 2-8: Summary Display

A



B



APPENDIX A

Protocol for Reverse Transcription and Amino-allyl Coupling of RNA

The following is a slight modification* of a protocol developed by Joe DeRisi (UCSF) and Rosetta Inpharmatics (Kirkland, WA). Original document can be obtained at www.microarrays.org.

A. RT Reaction

1. To anneal primer, mix 1-2 µg mRNA with 5 ug of anchored oligo-dT [(dT) 20 -VN] (Operon, HPLC purified) in a total volume of 18 µL. One reaction for sample mRNA and one for reference mRNA.

oligo dT	5µg of 2.5 µg/ µL	2µL
mRNA/water	1-2 µg	16 µL

2. Heat to 70C for 10 minutes. Cool on ice for 5 minutes.
3. Add 11.6 µL of nucleotide mix to each of Cy3 and Cy5 reactions.

Nucleotide Mix for one reaction

5X RT buffer		6.0 µL
50X dNTP stock solution		0.6
DTT	0.1M	3.0
Superscript II RT (Gibco)	200U/ µL	1.5
RNasin (Gibco, optional)	40U/ µL	0.5

50X dNTP stock solution using a 4:1 ratio aminoallyl-dUTP to dTTP***:

10 µL each 100 mM dATP, dGTP, dCTP (Pharmacia)
8µL 100 mM aminoallyl-dUTP** (Sigma, #A0410)
2µL 100 mM dTTP

**Dissolve 10 mg aminoallyl-dUTP in 170 µL water. Add approx. 6.8 µL 1N NaOH. Final pH is roughly 7.0 using pH paper.
***Altering the ratio of aminoallyl-dUTP to dTTP will affect the incorporation of Cy dye.

1X dNTP final concentration during labeling

500 µM each dATP, dCTP, dGTP
400 µM aminoallyl-dUTP
100 µM dTTP

4. Incubate reaction for 1 hour at 42C. Add additional 1 µL reverse transcriptase and continue incubation at 42C for an additional 1 hour.

B. Hydrolysis

1. Degrade RNA by addition of 15 µL of 0.1 N NaOH. Incubate at 70C for 10 minutes
2. Neutralize by addition of 15 µL 0.1 N HCl.

To continue with the amino-allyl dye coupling procedure, all Tris must be removed from the reaction to prevent the monofunctional NHS-ester Cy-dyes from coupling to free amine groups in solution.

3. Add 450 µL water to each reaction.

C. Cleanup

Add 500 µL neutralized, diluted reaction mix to a Microcon-30 filter (Amicon).

Spin at 12g for 7 minutes.

Discard flow through.

Repeat process two more times, refilling original filter with 450 µL water. Concentrate to 10 µL. Samples can be stored at -20C indefinitely.

D. Coupling

Add 0.5 µL 1M sodium bicarbonate, pH 9.0 to 50 mM final. Check 1M stock solution periodically for fluctuations in pH.

Monofunctional NHS-ester Cy3 (PA23001) and Cy5 dye (PA25001, Amersham) is supplied as a dry pellet. Each tube is sufficient to label 10 reactions under normal conditions. Dissolve dry pellet in 20 µL DMSO. Aliquot 2 µL into 10 single use tubes that are then dried in vacuo and store desiccated at 4C. NHS-ester conjugated Cy dye is rapidly hydrolyzed in water, therefore, do not store in DMSO or water. Decreasing the number of aliquots/dye tube may increase your signal.

If you have already made aliquots of dye, simply transfer your cDNA in bicarbonate buffer (10.5 µL) to the aliquot of dye. Alternatively, dissolve Cy dye in 10 ul DMSO and add 1 µL of dye to 10.5 µL of the cDNA reaction. 10% DMSO in the coupling reaction will not affect the chemical reaction. Aliquot unused dye and dry immediately.

Incubate 1 hour at RT in the dark. Mix every 15 minutes.

E. Quenching and Cleanup

Before combining Cy3 and Cy5 samples for hybridization, unreactive NHS-ester Cy dye must be quenched to prevent cross coupling.

Add 4.5 μL 4M hydroxylamine (Sigma).

Let reaction incubate 15 minutes in the dark.

To remove unincorporated/quenched Cy dyes, proceed with Qia-Quick PCR purification kit (QIAGEN). Method described below is as specified by manufacturer.

Combine Cy3 and Cy5 reactions.
Add 70 μL water.
Add 500 μL Buffer PB.
Apply to Qia-quick column and spin at 13K for 30-60 seconds. (optional: reapply flow-though for optimal binding).
Decant flow-through.
Add 750 μL Buffer PE and spin 30-60 seconds.
Decant flow-through.
Repeat PE wash two more times
Spin at high speed to dry column.
Transfer spin unit to fresh eppendorf tube.
Add 30 μL Buffer EB to center of filter and allow to sit 3 minutes at RT.
Spin at 13K rpm for 1 minute.
Repeat elution step again with another 30 μL of Buffer EB.
Pool eluates.

Add 420 TE and apply to fresh Microcon-30 filter.
Spin 12,000g to a volume of 29 μL or less.

For 38 μL array hybridization:

29 μL cDNA probe in TE
1μL polyA (10 μg; Sigma P9403)
1μL tRNA (10 μg; Gibco #15401-029)
7μL 20X SSC
1.2 μL SDS 10%

Heat to 100C for 2 minutes. Let stand 15 minutes RT.

Apply 38 μL to 40K array.
*Slight modifications to original protocol by Mitch Garber and Anatoly Urisman.

Preparation of Fluorescent DNA Probe from HUMAN mRNA or Total RNA using Direct Incorporation (Max Diehn/Ash Alizadeh protocol; 3/15/01) Modified for Yeast Hybridization

I. Preparing fluoresenctly labeled cDNA (probe):

To anneal primer, mix 2ug of mRNA or 50-100 g total RNA with 4ug of a regular or anchored oligo-dT primer in a total volume of 15.4 ul:

	Cy3	Cy5
mRNA (1 /)	x	y
Oligo-dT (4 g/l)	1	1
ddH 2O (DEPC)	to 15.4	to 15.4
Total volume:	15.4	15.4

(2 g of each if mRNA, 50-100 g if total RNA)
(Anchored: 5'-TTT TTT TTT TTT TTT TTT TTV N-3')

2. Heat to 65 oC for 10 min and cool on ice.
3. Add 14.6 mL of reaction mixture each to Cy3 and Cy5 reactions:

Reaction mixture	Microliters	Unlabelled dNTPs	Vol.	Final conc.
5X first-strand buffer*	6.0	dATP (100 mM)	25 uL	25 mM
0.1M DTT	3.0	dCTP (100 mM)	25 uL	25 mM
Unlabeled dNTPs	0.6	dGTP (100 mM)	25 uL	25 mM
Cy3 or Cy5 (1 mM, Amersham)	3.0	dTTP (100 mM)	10 uL	10 mM
Superscript II (200 U/uL, Gibco BRL)	2.0	ddH2O	15 uL	
Total volume:	14.6	Total volume:	100 uL	

* 5X first-strand buffer: 250 mM Tris-HCL (pH 8.3), 375mM KCl, 15mM MgCl2)

4. Incubate at 42 oC for 1 hr.
5. Add 1 lSSII (RT booster) to each sample. Incubate for an additional 0.5-1 hrs.
6. Degrade RNA and stop reaction by addition 15 ml of 0.1N NaOH, 2mM EDTA and incubate at 65-70 oC for 10 min. If starting with total RNA, degrade for 30 min instead of 10 min.
7. Neutralize by addition of 15 ml of 0.1N HCl.
8. Add 380 ml of TE (10mM Tris, 1mM EDTA) to a Microcon YM-30 column (Millipore). Next add the 60 ml of Cy5 probe and the 60 ml of Cy3 probe to the same microcon. (Note: If re-purification of cy dye flow-through is desired, do not combine probes until Wash 2.)
9. **WASH 1:** Spin column for 7-8 min. at 14,000 x g.
10. **WASH 2:** Remove flow-through and add 450 ul TE and spin for 7-8 min. at 14,000 x g. It is a good idea to save the flow trough for each set of reactions in a separate microcentrifuge tube in case Microcon membrane ruptures.
11. **WASH 3:** Remove flow-through and add 450 ul 1X TE and 20 g polyA RNA (10 g/ l, Sigma, #P9403). Spin 7-10 min. at 14,000 x g. Look for concentration of the probe in the microcon. The probe usually has a purple color at this point. Concentrate to a volume of less than or equal to the volume listed in the "Probe & TE" column in the table below. These low volumes are attained after the center of the membrane is dry and the probe forms a ring of liquid at the edges of the membrane. Make sure not to dry the membrane completely!
12. Invert the microcon into a clean tube and spin briefly at 14,000 RPM to recover the probe.

Cover Slip Size (mm)	Total Hyb Volume (ul)	Probe & TE (ul)	20x SSC (ul)	10% SDS (ul)
22 x 22	15	12	2.55	0.45
22 x 40	25	20	4.25	0.75
22 x 60	35	28	5.95	1.05

*20x SSC: 3.0 M NaCl, 300 mM NaCitrate (pH 7.0)

13. Adjust the probe volume to the value indicated in the "Probe & TE" column above.
14. For final probe preparation add 4.25 20XSSC and 0.75 10%SDS. When adding the SDS, be sure to wipe the pipette tip with clean, gloved fingers to rid of excess SDS. Avoid introducing bubbles and never vortex after adding SDS.
15. Denature probe by heating for 2 min at 100 oC, leave at 42C for 15-20 min and spin at 14,000 RPM.
16. Place the entire probe volume on the array under a the appropriately sized glass cover slip.
17. Hybridize at 65 oC for 14 to 18 hours in a custom slide chamber with humidity maintained by a small reservoir of 3X SSC (spot around 3-6 3X SSC at each corner of the slide, as far away from the array as possible).

II. Washing and Scanning Arrays:

1. Ready washes in 250 ml chambers to 200 ml volume as indicated in the table below. Avoid adding excess SDS. The Wash 1A chamber and the Wash 2 chambers should each have a slide rack ready. All washes are done at room temperature.
2.

Wash	Description	Vol (ml)	SSC	SDS (10%)
1A	2x SSC, 0.03% SDS	200	200 ml 2x	0.6 ml
1B	2x SSC	200	200 ml 2x	--
2	1x SSC	200	200 ml 1x	--
3	0.2x SSC	200	200 ml 0.2x	--

3. Blot dry chamber exterior with towels and aspirate any remaining liquid from the water bath.
4. Unscrew chamber; aspirate the holes to remove last traces of water bath liquid.
5. Place arrays, singly, in rack, inside Wash I chamber (maximum 4 arrays at a time). Allow cover slip to fall, or *carefully* use forceps to aid cover slip removal if it remains stuck to the array. DO NOT AGITATE until cover slip is safely removed. Then agitate for 2 min.
6. Remove array by forceps, rinse in a Wash II chamber *without* a rack, and transfer to the Wash II chamber with the rack. This step minimizes transfer of SDS from Wash I to Wash II.
7. Wash arrays by submersion and agitation for 2 min in Wash II chamber, then for 2 min in Wash III (transfer the entire slide rack this time).
8. Spin dry by centrifugation in a slide rack in a Beckman GS-6 tabletop centrifuge at 600 RPM for 2 min
9. Scan arrays immediately.

References

Aon, J. C. and S. Cortassa (2001). "Involvement of nitrogen metabolism in the triggering of ethanol fermentation in aerobic chemostat cultures of Saccharomyces cerevisiae." Metab Eng **3**(3): 250-64.

Arreguin de Lorencez, M. and O. Kappeli (1987). "Regulation of gluconeogenic enzymes during the cell cycle of Saccharomyces cerevisiae growing in a chemostat." J Gen Microbiol **133 (Pt 9)**: 2517-22.

Ashburner, M., C. A. Ball, et al. (2000). "Gene ontology: tool for the unification of biology. The Gene Ontology Consortium." Nat Genet **25**(1): 25-9.

Auberson, L. C., C. V. Ramseier, et al. (1989). "Further evidence for the existence of a bottleneck in the metabolism of Saccharomyces cerevisiae." Experientia **45**(11-12): 1013-8.

Baganz, F., A. Hayes, et al. (1998). "Quantitative analysis of yeast gene function using competition experiments in continuous culture." Yeast **14**(15): 1417-27.

Birol, G., A. M. Zamamiri, et al. (2000). "Frequency analysis of autonomously oscillating yeast cultures." **35**(10): 1085-1091.

Boer, V. M., J. H. de Winde, et al. (2003). "The genome-wide transcriptional responses of Saccharomyces cerevisiae grown on glucose in aerobic chemostat cultures limited for carbon, nitrogen, phosphorus, or sulfur." J Biol Chem **278**(5): 3265-74.

Boraas, M. E., D. B. Seale, et al. (1998). "Rotifer size distribution changes during transient phases in open cultures." Hydrobiologia **387/388**: 477-482.

Buziol, S., J. Becker, et al. (2002). "Determination of in vivo kinetics of the starvation-induced Hxt5 glucose transporter of Saccharomyces cerevisiae." FEM Yeast Res **2**(3): 283-91.

Castrillo, J. I. and U. O. Ugalde (1994). "A general model of yeast energy metabolism in aerobic chemostat culture." Yeast **10**(2): 185-97.

Cazzador, L. (1991). "Analysis of oscillations in yeast continuous cultures by a new simplified model." Bull Math Biol **53**(5): 685-700.

Chen, P. S., T. T. Y., et al. (1956). "Microdetermination of Phosphorus." Analytical Chemistry **28**: 1756-1758.

Crabtree, H. G. (1929). "Observations on the carbohydrate metabolism of tumors." Biochemical Journal **23**: 536.

de Kock, S. H., J. C. du Preez, et al. (2000). "The effect of vitamins and amino acids on glucose uptake in aerobic chemostat cultures of three Saccharomyces cerevisiae strains." Syst Appl Microbiol **23**(1): 41-6.

DeRisi, J. L., V. R. Iyer, et al. (1997). "Exploring the metabolic and genetic control of gene expression on a genomic scale." Science **278**(5338): 680-6.

Diderich, J. A., L. M. Raamsdonk, et al. (2002). "Effects of a hexokinase II deletion on the dynamics of glycolysis in continuous cultures of Saccharomyces cerevisiae." FEM Yeast Res **2**(2): 165-72.

du Preez, J. C., S. H. de Kock, et al. (2000). "The relationship between transport kinetics and glucose uptake by Saccharomyces cerevisiae in aerobic chemostat cultures." Antonie Van Leeuwenhoek **77**(4): 379-88.

Dunham, M. J., H. Badrane, et al. (2002). "Characteristic genome rearrangements in experimental evolution of Saccharomyces cerevisiae." Proc Natl Acad Sci U S A **99**(25): 16144-9.

Dwight, S. S., M. A. Harris, et al. (2002). "Saccharomyces Genome Database (SGD) provides secondary gene annotation using the Gene Ontology (GO)." Nucleic Acids Res **30**(1): 69-72.

Egli, T., U. Lendenmann, et al. (1993). "Kinetics of microbial growth with mixtures of carbon sources." Antonie Van Leeuwenhoek **63**(3-4): 289-98.

Eisen, M. B., P. T. Spellman, et al. (1998). "Cluster analysis and display of genome-wide

expression patterns." Proc Natl Acad Sci U S A **95**(25): 14863-8.
Evans, C. T. and C. Ratledge (1983). "A comparison of the oleaginous yeast, Candida curvata, grown on different carbon sources in continuous and batch culture." Lipids **18**(9): 623-9.
Fiaux, J., Z. P. Cakar, et al. (2003). "Metabolic-Flux Profiling of the Yeasts Saccharomyces cerevisiae and Pichia stipitis." Eukaryot Cell **2**(1): 170-80.
Flikweert, M. T., M. Kuyper, et al. (1999). "Steady-state and transient-state analysis of growth and metabolite production in a Saccharomyces cerevisiae strain with reduced pyruvate-decarboxylase activity." Biotechnol Bioeng **66**(1): 42-50.
Gasch, A. P., P. T. Spellman, et al. (2000). "Genomic expression programs in the response of yeast cells to environmental changes." Mol Biol Cell **11**(12): 4241-57.
Gasch, A. P. and M. Werner-Washburne (2002). "The genomics of yeast responses to environmental stress and starvation." Funct Integr Genomics **2**(4-5): 181-92.
Gollub, J., C. A. Ball, et al. (2003). "The Stanford Microarray Database: data access and quality assessment tools." Nucleic Acids Res **31**(1): 94-6.
Gombert, A. K., M. Moreira dos Santos, et al. (2001). "Network identification and flux quantification in the central metabolism of Saccharomyces cerevisiae under different conditions of glucose repression." J Bacteriol **183**(4): 1441-51.
Hayes, A., N. Zhang, et al. (2002). "Hybridization array technology coupled with chemostat culture: Tools to interrogate gene expression in Saccharomyces cerevisiae." Methods **26**(3): 281-90.
Herrero, P., R. Fernandez, et al. (1985). "Differential sensitivities to glucose and galactose repression of gluconeogenic and respiratory enzymes from Saccharomyces cerevisiae." Arch Microbiol **143**(3): 216-9.
Huisman, J., H. C. Matthijs, et al. (2002). "Principles of the light-limited chemostat: theory and ecological applications." Antonie Van Leeuwenhoek **81**(1-4): 117-33.
Ishige, T., M. Krause, et al. (2003). "The Phosphate Starvation Stimulon of Corynebacterium glutamicum Determined by DNA Microarray Analyses." J Bacteriol **185**(15): 4519-29.
Joy, B., G. Steele, et al. (2000). Java(TM) Language Specification, Addison-Wesley Publishing Company.
Kellis, M., N. Patterson, et al. (2003). "Sequencing and comparison of yeast species to identify genes and regulatory elements." Nature **423**(6937): 241-54.
Kiers, J., A. M. Zeeman, et al. (1998). "Regulation of alcoholic fermentation in batch and chemostat cultures of Kluyveromyces lactis CBS 2359." Yeast **14**(5): 459-69.
Kirk, N. and P. W. Piper (1994). "Growth rate influences MF alpha 1 promoter activity in MAT alpha Saccharomyces cerevisiae." Appl Microbiol Biotechnol **42**(2-3): 340-5.
Kubitscheck, H. E. (1970). Introduction to Research with Continuous Cultures. Englewood Cliffs, N. J., Prentice-Hall, Inc.
Lange, P. and P. E. Hansche (1980). "Mapping of a centromere-linked gene responsible for constitutive acid phosphatase synthesis in yeast." Mol Gen Genet **180**(3): 605-7.
Larsson, C., U. von Stockar, et al. (1993). "Growth and metabolism of Saccharomyces cerevisiae in chemostat cultures under carbon-, nitrogen-, or carbon- and nitrogen-limiting conditions." J Bacteriol **175**(15): 4809-16.
Lemoigne, M., J. P. Aubert, et al. (1954). "La production d'alcool et la rendement de croissance de la levure de boulangerie cultivee en aerobiose." Ann. Inst.Pasteur **87**: 427-433.
Leuenberger, H. G. (1971). "Cultivation of Saccharomyces cerevisiae in continuous culture. I. Growth kinetics of a respiratory deficient yeast strain grown in continuous culture." Arch Mikrobiol **79**(2): 176-86.
Leuenberger, H. G. (1972). "Cultivation of Saccharomyces cerevisiae in continuous culture. II. Influence of the crabtree effect on the growth characteristics of Saccharomyces cerevisiae grown in a glucose limited chemostat." Arch Mikrobiol **83**(4): 347-58.
Lyons, T. J., A. P. Gasch, et al. (2000). "Genome-wide characterization of the Zap1p zinc-responsive regulon in yeast." Proc Natl Acad Sci U S A **97**(14): 7957-62.
McNair, J. N., M. E. Boraas, et al. (1998). "Size-structure dynamics of the rotifer chemostat: a

simple physiologically structured model." Hydrobiologia **387/388**: 469-476.
Monod, J. (1942). Recherches sure la croissance des cultures bacteriennes. Paris, Herman & Cie.
Monod, J. (1950). "La Technique de culture continue. Theorie et applications." Ann. Inst. Pasteur **79**: 390-410.
Novick, A. and L. Szilard (1950). "Description of the chemostat." Science: 715-716.
Novick, A. and L. Szilard (1950). "Experiments with the chemostat on spontaneous mutations of bacteria." Proc Natl Acad Sci U S A **36**: 708-719.
Ogawa, N., J. DeRisi, et al. (2000). "New components of a system for phosphate accumulation and polyphosphate metabolism in Saccharomyces cerevisiae revealed by genomic expression analysis." Mol Biol Cell **11**(12): 4309-21.
Olz, R., K. Larsson, et al. (1993). "Energy flux and osmoregulation of Saccharomyces cerevisiae grown in chemostats under NaCl stress." J Bacteriol **175**(8): 2205-13.
Overkamp, K. M., P. Kotter, et al. (2002). "Functional analysis of structural genes for NAD(+)-dependent formate dehydrogenase in Saccharomyces cerevisiae." Yeast **19**(6): 509-20.
Padilla, P. A., E. K. Fuge, et al. (1998). "The highly conserved, coregulated SNO and SNZ gene families in Saccharomyces cerevisiae respond to nutrient limitation." J Bacteriol **180**(21): 5718-26.
Pahlman, I. L., L. Gustafsson, et al. (2001). "Cytosolic redox metabolism in aerobic chemostat cultures of Saccharomyces cerevisiae." Yeast **18**(7): 611-20.
Paredes-Lopez, O., E. Camargo-Rubio, et al. (1976). "Influence of specific growth rate on biomass yield, productivity, and compostion of Candida utilis in batch and continuous culture." Appl Environ Microbiol **31**(4): 487-91.
Peter Smits, H., J. Hauf, et al. (2000). "Simultaneous overexpression of enzymes of the lower part of glycolysis can enhance the fermentative capacity of Saccharomyces cerevisiae." Yeast **16**(14): 1325-34.
Piper, M. D., P. Daran-Lapujade, et al. (2002). "Reproducibility of oligonucleotide microarray transcriptome analyses. An interlaboratory comparison using chemostat cultures of Saccharomyces cerevisiae." J Biol Chem **277**(40): 37001-8.
Planta, R. J. and W. H. Mager (1998). "The list of cytoplasmic ribosomal proteins of Saccharomyces cerevisiae." Yeast **14**(5): 471-7.
Pollack, J. R., C. M. Perou, et al. (1999). "Genome-wide analysis of DNA copy-number changes using cDNA microarrays." Nat Genet **23**(1): 41-6.
Pretorius, I. S. (2000). "Tailoring wine yeast for the new millennium: novel approaches to the ancient art of winemaking." Yeast **16**(8): 675-729.
Salusjarvi, L., M. Poutanen, et al. (2003). "Proteome analysis of recombinant xylose-fermenting Saccharomyces cerevisiae." Yeast **20**(4): 295-314.
Sarvari Horvath, I., C. J. Franzen, et al. (2003). "Effects of Furfural on the Respiratory Metabolism of Saccharomyces cerevisiae in Glucose-Limited Chemostats." Appl Environ Microbiol **69**(7): 4076-86.
Stephanopoulos, G., D. Hwang, et al. (2002). "Mapping physiological states from microarray expression measurements." Bioinformatics **18**(8): 1054-63.
Stuckrath, I., H. C. Lange, et al. (2002). "Characterization of null mutants of the glyoxylate cycle and gluconeogenic enzymes in S. cerevisiae through metabolic network modeling verified by chemostat cultivation." Biotechnol Bioeng **77**(1): 61-72.
ter Linde, J. J., H. Liang, et al. (1999). "Genome-wide transcriptional analysis of aerobic and anaerobic chemostat cultures of Saccharomyces cerevisiae." J Bacteriol **181**(24): 7409-13.
Thomas, D. and Y. Surdin-Kerjan (1997). "Metabolism of sulfur amino acids in Saccharomyces cerevisiae." Microbiol Mol Biol Rev **61**(4): 503-32.
van de Peppel, J., P. Kemmeren, et al. (2003). "Monitoring global messenger RNA changes in externally controlled microarray experiments." EMBO Rep **4**(4): 387-93.
van den Berg, M. A., P. de Jong-Gubbels, et al. (1998). "Transient mRNA responses in

chemostat cultures as a method of defining putative regulatory elements: application to genes involved in Saccharomyces cerevisiae acetyl-coenzyme A metabolism." Yeast **14**(12): 1089-104.

van Hoek, P., J. P. van Dijken, et al. (2000). "Regulation of fermentative capacity and levels of glycolytic enzymes in chemostat cultures of Saccharomyces cerevisiae." Enzyme Microb Technol **26**(9-10): 724-736.

van Maris, A. J., B. M. Bakker, et al. (2001). "Modulating the distribution of fluxes among respiration and fermentation by overexpression of HAP4 in Saccharomyces cerevisiae." FEM Yeast Res **1**(2): 139-49.

Van Uden, N. and A. Madeira-Lopes (1975). "Dependence of the maximum temperature for growth of Saccharomyces cerevisiae on nutrient concentration." Arch Microbiol **104**(1): 23-8.

Van Urk, H., P. R. Mak, et al. (1988). "Metabolic responses of Saccharomyces cerevisiae CBS 8066 and Candida utilis CBS 621 upon transition from glucose limitation to glucose excess." Yeast **4**(4): 283-91.

Vrana, D. (1976). "Daughter cells as an important factor in determining the physiological state of yeast populations." Biotechnol Bioeng **18**(3): 297-309.

Wahlbom, C. F., R. R. Cordero Otero, et al. (2003). "Molecular analysis of a Saccharomyces cerevisiae mutant with improved ability to utilize xylose shows enhanced expression of proteins involved in transport, initial xylose metabolism, and the pentose phosphate pathway." Appl Environ Microbiol **69**(2): 740-6.

Weusthuis, R. A., J. T. Pronk, et al. (1994). "Chemostat cultivation as a tool for studies on sugar transport in yeasts." Microbiol Rev **58**(4): 616-30.

Xu, R. and X. Li (2003). "A comparison of parametric versus permutation methods with applications to general and temporal microarray gene expression data." Bioinformatics **19**(10): 1284-9.

Ye, L., J. A. Berden, et al. (2001). "Expression and activity of the Hxt7 high-affinity hexose transporter of Saccharomyces cerevisiae." Yeast **18**(13): 1257-67.