University of Tennessee, Knoxville

TRACE: Tennessee Research and Creative Exchange

Chancellor's Honors Program Projects

Supervised Undergraduate Student Research and Creative Work

Fall 12-2001

Developing a Software Tool for Creating Interactive Electronic Forms Using the Spreadsheet Paradigm

David Robert Resseguie
University of Tennessee-Knoxville

Follow this and additional works at: https://trace.tennessee.edu/utk_chanhonoproj

Recommended Citation

Resseguie, David Robert, "Developing a Software Tool for Creating Interactive Electronic Forms Using the Spreadsheet Paradigm" (2001). *Chancellor's Honors Program Projects.*
https://trace.tennessee.edu/utk_chanhonoproj/490

This is brought to you for free and open access by the Supervised Undergraduate Student Research and Creative Work at TRACE: Tennessee Research and Creative Exchange. It has been accepted for inclusion in Chancellor's Honors Program Projects by an authorized administrator of TRACE: Tennessee Research and Creative Exchange. For more information, please contact trace@utk.edu.

Developing a Software Tool for Creating Interactive Electronic Forms Using the Spreadsheet Paradigm

Prepared By: David Resseguie

Faculty Advisor: Dr. Brad VanderZanden

Honor Program Senior Project

December 13, 2001

ABSTRACT

This project was completed in conjunction with Dr. VanderZanden's ongoing research into spreadsheet programming and computer user interfaces. The project is centered on creating a tool that will make it easy to create interactive electronic forms using the spreadsheet paradigm. The basic idea is to allow the developer to create a freeform table in which any cell can compute its value using a formula, much like users are accustomed to doing in applications like Microsoft Excel. The table can be adapted to look like a form and all types of interactive widgets or icons can be placed into the cells. This tool will allow fast development of electronic forms with the flexibility, power, and simplicity of spreadsheet programming. Java is being used in the development of this tool for maximum portability. The complete tool will be very complex, so this project concentrates on one particular aspect of the application, that being the functionality for creating the form layouts. There are several issues addressed by this project. The first issue is the actual drawing of the table, such as the user interface and drawing tools. Second, the user input must be checked and corrected. The third issue deals with converting the individually drawn lines to cells using pattern recognition, interpolation, and error checking. The code for this project will later be incorporated into Dr. VanderZanden's work to add the rest of the application's functionality.

INTRODUCTION

This project was completed under the guidance of Dr. Brad VanderZanden, an associate professor in the Department of Computer Science at the University of Tennessee. Dr. VanderZanden's research centers around spreadsheet programming and computer user interfaces. As part of his research, he is seeking to develop a software tool for easily creating interactive electronic forms. All programming for this tool will be done using the Java programming language to insure maximum portability across platforms. The goal of this project is to implement the functionality for creating the form layouts, a key piece of the completed application. The primary focus of the project is to produce powerful and flexible code that can later be incorporated into Dr. VanderZanden's existing work. This paper describes the purpose of the project, gives reasoning behind the technical decisions, and discusses the skills that were applied to complete the project.

The first section discusses the problems companies face when trying to switch to paperless operation and how the proposed software tool is intended to overcome these obstacles. This section also gives details on the actual requirements of this project for handling the user input and the approach taken to handle each requirement. The second section of this paper gives a technical description of the project. The code organization is explained along with reasons for the various decisions that were made. Principles of object-oriented design are only briefly discussed and a general programming knowledge is expected of the reader. The next section discusses the skills used in producing this project and makes note of experiences and lessons learned. The final section gives recommendations for future enhancements and improvements to the software.

STATEMENT OF PROBLEM AND PROPOSED SOLUTION

The high cost of document handling and storage has caused many companies to push for "paperless" operation. The new strategy seeks to eliminate the costly and time-consuming processing of hardcopy documents by entering data directly into electronic format. For most companies this requires converting many traditional paper forms to a usable electronic format. There are several obstacles to be considered for this conversion. First, developing such forms requires hiring expensive computer programmers. Producing electronic forms requires special knowledge in creating graphical user interfaces and programmers with these skills are in high demand. Only large firms and organizations have the resources to employ such programmers. A second obstacle is the lack of flexibility of these custom designed forms. Each time a new form is needed, a programmer must be contracted to design and implement the new electronic form. It is also difficult to publish the same form in different formats. For example, a company may want to make their form available to customers via the web or as a standalone application for use in-house.

Dr. VanderZanden's proposed solution to this problem is to develop a software tool for quickly creating interactive electronic forms. The tool will provide a user interface suitable for intermediate computer users. Some of the key features include a standard drawing interface for sketching the form layout, drag-and-drop functionality for form elements, spreadsheet style formula programming, and flexible publishing methods. A company desiring to convert to paperless operation would only need to purchase the software tool to enable a moderately trained and relatively inexpensive office assistant to produce the electronic forms. Once designed, the form could be published in multiple formats to easily suit the needs of the company. There are tools available on the market that have subsets of this functionality. Many of these applications either do not provide some of the key features as described above or are too difficult for an

intermediate computer user. This new tool seeks to combine the power and flexibility of spreadsheet programming with ease of use.

The development of a software application as described above is a task usually undertaken by a team of programmers. Because of time constraints, this project was limited to implementing a subset of the functionality. The focus is directed toward the functionality related to handling the user input and creating the form layout. The implementation of the project was divided into three tasks. The technical details will be for each of these areas will be covered in the next section.

The first task is the actual drawing of the form layout. A good amount of consideration went into designing this section of the application. The goal is to produce an intuitive drawing interface that will not require a large learning curve. The project implements a very usable interface without confusing aspects like moded operation. The user is simply required to sketch the rectangular layout of the form. For creating lines, the user draws with the mouse while holding down the left mouse button. The right mouse button is used for moving nodes and straightening lines.

Second, the user input must be checked and corrected. To properly recognize individual cells, all lines must be either horizontal or vertical and all nodes must have at least two lines attached to them. Lines and nodes that do not meet these requirements are highlighted and brought to the user's attention. The application also provides a "snapping" mechanism to simplify the line straightening process. This feature avoids many of the annoyances of standard snap-to-grid options and simply straightens lines when they are within a customizable threshold of horizontal or vertical.

The third task deals with converting the individually drawn lines to cells using pattern recognition, recursion, and error checking. After finishing the form layout, the user clicks a button to begin the cell recognition process. There is no user interaction during the calculations. The algorithm for recognizing cells is described in more detail in the technical section of this paper. Because the project is only a piece of the complete software tool, after the cells are calculated they are simply stored internally and a count is returned for verification purposes. As part of the complete application, the cells could be accessed for further modification, identification, and formula programming.

An additional goal for the project is to gain experience in good object-oriented design, including a flexible and powerful interface. Such a design allows for easier maintenance in the future by other programmers. As the full application is developed, there will certainly be new ideas and features to incorporate into the existing design. A well-designed application allows for easy modification and addition of features. All these factors were considered in the development of this project.

TECHNICAL DESCRIPTION

The Java programming language was used for the implementation of this project because of its object-oriented design, available graphical user interface (GUI) API, and maximum portability between platforms. All classes for the project are stored in a package named com.resseguie.form, ensuring no naming conflicts with other classes to be used in the final application. The Model-View-Controller (MVC) architecture was used in the design of this project. The data is stored independently from the view, using the MVC design principles, to allow maximum expandability in the future.

The FormEditorModel class serves as the project's model. This class contains all methods needed for adding and removing elements, retrieving lists of elements, and managing the internal data structures. FormEditorSketchView is the class for the drawing panel view and controller. This class handles all the user's drawing input and passes along the appropriate commands to the model. The FormEditorConstants interface is used for application wide variables and defaults. The interface allows simple modification of variables without having to use error prone search and replace functions. FormEditorLine and FormEditorNode classes were developed to simplify and modularize interaction with elements on the screen. Each of these classes implements the DrawableFormElement interface for use in a single UpdateAction class for communication between the model and the view. The FormEditorMoveInteractor is a subclass of SilMoveInteractor that adds functionality to do extra processing after a node has been moved to recognize new intersections and perform snapping functions. The Cell class is used to represent individually recognized cells of the form. This class will certainly be modified or sub-classed for the full application.

For drawing and manipulating elements in the drawing panel, the silhouette graphics package is utilized. Silhouette was designed by Dr. VanderZanden to simplify custom graphics drawing

within a GUI. It also serves as a display manager for the application. These strengths are applied to produce cleaner and more manageable code for the project. The FormEditorLine and FormEditorNode classes each have methods for returning an instance of a SilShape representative of their location for use with silhouette. Use of the silhouette interactors model also simplifies drawing feedback objects during the drawing phase. Silhouette constraints are used to maintain line positions when the user moves a node.

The most difficult programming task is handling node and line intersections during the drawing phase. These calculations are performed when a line is initially created and when editing the position of a line. Some of the necessary geometric shape calculations are included in the Java2D API. Each FormEditorNode and FormEditorLine object stores an instance of an Ellipse2D and Line2D respectively for use in these calculations. Custom methods were written to make calculations not included with the standard Java interface. For example, the Line2D class provides a method for determining if two lines intersect, but there is no method for determining the intersection point. Therefore a getLineIntersection(...) method was included in the FormEditorLine class to perform that calculation. These custom methods also operate on the Ellipse2D and Line2D objects and use standard rules of shape geometry.

There are multiple intersection cases to consider. The first case is when a node overlaps another node. When two nodes are very close together it is assumed that the user intended for both lines to connect at that point. The second node is removed and all lines associated with that node are transferred to the first. The second case to consider is when a node is placed on top of an existing line or a line is drawn across an existing node. In this case the line is broken into two segments at the location of the node. This task is accomplished by first removing the original line, then a new line is added from each of the original endpoints to the new intersection point. Handling line intersections is the final case to consider. If it is determined that two lines intersect, the

previously mentioned getLineIntersection(...) method is called and the returned Point2D object is used to create a new FormEditorNode object at the intersection point. Each of the lines is then segmented as described above for line-node intersections. Handling single instances of each of these cases is not extremely complex. The difficulty comes when multiple intersections have to be calculated at once. Careful manipulation of data structures and recursive method calls are used to accomplish these tasks.

Many helper functions are included in the FormEditorNode and FormEditorLine classes to provide cleaner modularized code. Some of these methods include isHorizontal(), getOtherEndpoint(...), hasNoLines(). All of the information gathered from these methods could be calculated as needed, but good object-oriented design dictates that common functionality be factored out and included in the underlying class. This coding style helped tremendously in the debugging stage for this project.

A recursive algorithm is also used for converting the individually drawn lines into cells. The FormEditorNode class has a method to return the next node in each direction: north, south, east, and west. The class also has methods for returning the next Northeast, Northwest, Southeast, and Southwest nodes. For example, the getNorthEastNode() method returns the next node to the east that has a line extending toward the south. Each of these methods run recursively until the proper node is found. If the algorithm is unable to calculate the four nodes of the cell, a CellRecognitionException is thrown. Under normal circumstances this exception will only be raised if a non-rectangular area is found. The affected nodes are shaded so the user can correct the problem before trying again. Properly recognized cells are stored in a data structure in the model for later retrieval and manipulation.

SKILLS APPLIED

This project involves many different disciplines of computer science. These topics have been covered in computer science classes at the University of Tennessee, but this project gave the opportunity to combine them and gain experience with how they interact with one another. The main benefit from this project is the experience in designing and implementing a completely object-oriented graphical application. The project was an excellent exercise in writing modular and maintainable code. The benefits of this style code were evident in the debugging stage of the project. For example, after adding a new piece of functionality in the FormEditorSketchView class, it began behaving strangely. Several rounds of testing indicated that there was a fundamental flaw in the usage of one of the data structures. Instead of having to search the code for every instance of the problem, only one change was needed in a low-level function and the entire problem was corrected. The code is also written so that the underlying data structures can be changed without affecting the behavior of the high-level classes. The project also gave experience with another aspect of object-oriented design, Model-View-Controller architecture. Although course instruction covers the theory behind MVC, this project provided the opportunity to apply the principle to a real world application.

The experience in creating a graphical user interface is also invaluable. The project presented the need to consider many various issues. The first decision was to use silhouette as the display manager. Silhouette is a very powerful tool and the skills learned from this project will be very applicable in the future. The various geometric shape calculations also relate directly to topics covered in computer science courses. Often times such formulas are difficult to understand until actually implemented. Writing the custom functions to handle these calculations led to a better understanding of the mathematics behind computer graphics. Computer-user interaction is another important topic in computer science. Developing a GUI provides experience in designing intuitive interfaces as well as the actual implementation.

RECOMMENDATIONS FOR FUTURE WORK

As usual in software design, there are areas of this application that can be improved. The following are a few recommendations for future study and enhancements. One possibility is to enhance the underlying data structures. The project currently uses Java's built-in LinkedList class for storing nodes, lines, and cells. Java has an extensive set of available data structures based on the Collections class. Tests should be performed to compare these and other custom data structures to determine which performs the best for this application.

General improvements to the user interface would also be advised. Additional feedback operations would make the application more intuitive for the intermediate computer user. The completed application also needs help functions to describe the drawing process. Research should also be done for a more intuitive method for deleting nodes and lines from the sketch view.

If a user designs a very complex form layout, some of the intersection methods begin to run slightly slower because the entire node and line data structures must be traversed for each calculation. Aside from using a more efficient data structure, a variation of a quad-tree could be used to limit the number of comparisons to be made. This algorithm would perform much like a quad-tree implementation for a display manager where only those objects lying within the same section of the view are acted upon.

This project will serve as a strong foundation for the complete software tool envisioned by Dr. VanderZanden. The modular design makes the above mentioned enhancements relatively easy. Interaction with Dr. VanderZanden will continue to ensure that this project can be seamlessly integrated into the final application.

Appendix E - UNIVERSITY HONORS PROGRAM SENIOR PROJECT - APPROVAL

Name: David Robert Resseguie

College: Arts and Sciences Department: Computer Science & Math

Faculty Mentor: Brad Vander Zanden

PROJECT TITLE: Developing a Software Tool for Creating Interactive Electronic Forms Using the Spreadsheet Paradigm.

I have reviewed this completed senior honors thesis with this student and certify that it is a project commensurate with honors level undergraduate research in this field.

Signed: Brad Vander Zanden, Faculty Mentor

Date: 12/13/01

General Assessment - please provide a short paragraph that highlights the most significant features of the project.

Comments (Optional):