To the Graduate Council:

I am submitting herewith a thesis written by Eric Freeman Mann entitled "Mathematical Representations of the Architecture of Graphical User Interfaces." I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Mathematics.

Kenneth R. Kimble

Major Professor

We have read this thesis and
recommend its acceptance:

Donald J. Malloy

K.C. Reddy

Bruce A. Whitehead

Acceptance for the Council:

Anne Mayhew

Vice Provost and Dean of
Graduate Studies

(Original signatures are on file with official student records.)

MATHEMATICAL REPRESENTATIONS OF THE ARCHITECTURE OF GRAPHICAL USER INTERFACES

A Thesis Presented
for the
Master of Science Degree
The University of Tennessee, Knoxville

Eric Freeman Mann
August 2004

ACKNOWLEDGMENTS

This thesis would not have been possible without the patience and commitment of my advisor, Dr. Ken Kimble, and my committee, Dr. Don Malloy, Dr. K.C. Reddy, and Dr. Bruce Whitehead. Their willingness to engage rigorously in the development of this work has been integral to its success. Dr. Kimble's guidance was key to keeping the waters between computer science and mathematics significantly clearer.

Thanks are due to Dr. Boris Kupershmidt and Jeffrey Rogers for their contribution of grammatical and stylistic corrections. Dr. Kupershmidt's mathematical eye has been a generous gift to the development of this text.

The generosity, encouragement and understanding of my wife, Laurie Zimmerman Mann, have been a godsend. Without Laurie's inspiration and support, there is little chance I would have undertaken a graduate education.

ABSTRACT

This text is an introduction to mathematical representations of graphical user interfaces, GUIs. Vectors are employed as a means to represent the state of a GUI and the user interaction with a GUI. These representations form a model of the behavior of a GUI over time. The usefulness of the model in testing and developing well-behaved GUIs is discussed and demonstrated by example.

TABLE OF CONTENTS

LIST OF FIGURES

NOMENCLATURE

a_i	access vector of ON/OFF access-states of GUI controls at time i
a_i^k	k^{th} component of the vector a_i corresponding to the k^{th} control
c_i	condition vector at time i
d_i	vector of GUI user data at time i
d_i^k	k^{th} component of the vector d_i
Γ	set of transformation sequences that can be completed with a GUI
m	number of events in a complete transformation sequence
n	number of user interface controls in a GUI
r_k	type of the k^{th} control
$sd(r_k)$	set of possible user data contained in control of type r_k
$su(r_k)$	set of possible user interactions with a control of type r_k
$T(u_i, c_i)$	procedure, as a transformation, executed in response to u_i
$\mathbb{T}_m$	set of transformation sequences with m events
u_i	vector of user interactions at time i
u_i^k	k^{th} component of the vector u_i corresponding to the k^{th} control

CHAPTER 1
INTRODUCTION AND GENERAL INFORMATION

Research Questions on Graphical User Interfaces

This work began as an inquiry into design principles for graphical user interfaces, GUIs. Some of the questions asked at the outset were: "What does it take for a GUI to be robust and easy to use?", "What does it mean for a GUI to be easy to use?", "How useful is it to have a GUI behave as a omnipresent expert?", and "In what ways does the psychology of the user play a role in the design and implementation of a GUI?".

People researching graphical user interfaces are addressing and answering many of these questions. Virvou and Kabassi have been actively researching questions about the performance of "intelligent graphical user interfaces", in [$VKa2003$] and [$VKa2004$]. Kollar of the University of Debrecen, in [$Kol2003$], proposes a graphical user interface markup language, GUIML, "an XML-based language", to improve platform and language independence as well as improve the efficiency of GUI development. In his article, *Semiotics and GUI Design*, Callahan considers the question of what a user needs to know from the perspective of semiotics, "the study of signs " [$Cal1994$]. There is even a document, [$Hob1995$], providing "the cardinal dos and don'ts" of GUI design.

Motivation for this Research

Questions guiding this research of graphical user interfaces began moving toward the subject of mathematical modeling. "How would one generalize the notion of a GUI?", "What mathematics could be used to represent what a GUI is and does?", and ultimately, "Is there a way to model the structure and behavior of a GUI?".

This work was motivated by a desire to have a useful means to model the structure and behavior of graphical user interfaces. A key to the development and testing of well-behaved GUIs will be seen to be the ability to track the possible sequences of user interactions and catch any unexpected or unwanted behavior. It was desired to establish a model that would provide these capabilities.

Conventions Used in this Text

Italics

Important words accompanied by definitions will be *italicized* when they first appear. Mathematical quantities will be italicized as well.

Subscripts and Superscripts

The letters i and j are reserved for subscripts and are used to denote time. The letters k and l are reserved for superscripts and are used to reference GUI controls.[1] A superscript T refers to the transpose.

Equations

Equations are numbered consecutively throughout the text. Equations are referenced by their equation number, #, either as "Equation #" or "(#)".

Preliminary Definitions

Graphical User Interface

According to the MATLAB® Help Documentation, [*Mat*2004],

> By providing an interface between the user and the application's underlying code, GUIs enable the user to operate the application without knowing the commands [which] would be required by a command line interface. For this reason, applications that provide GUIs are easier to learn and use than those that are run from the command line.

While this is not exactly a definition, it provides a sense of what GUIs are intended to do.

Alhir provides this definition: "A *system* is a collection of elements organized for a specific purpose" [*Alh*2003]. The elements in a graphical user interface are the objects with which a user may interact. In this text, a GUI is viewed as a system for transforming input through a series of user interactions into output.

Architecture

As described in [*Alh*2003], the elements of a system and their interaction are a system's architecture. The elements are a system's *structure*, and the actions and interactions of elements are a system's *behavior*.

[1] Refer to the Nomenclature on page vii for a preview of this usage.

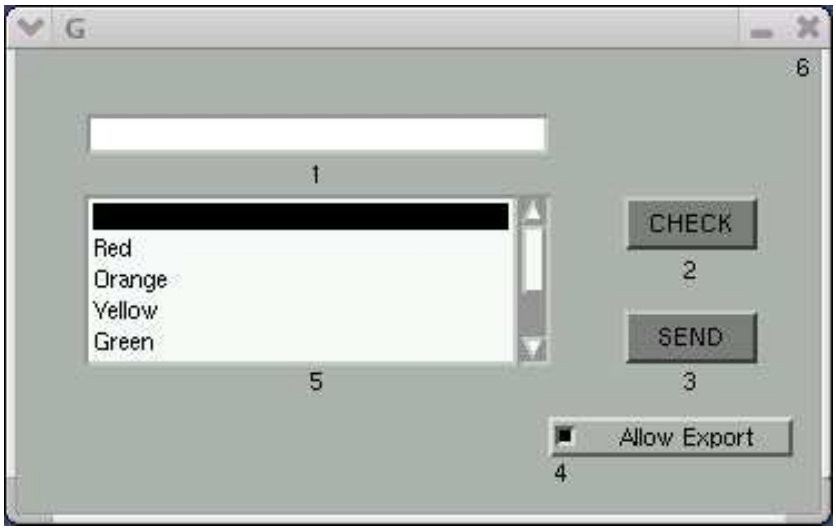

Figure 1: An Example of a GUI.

Control

A control, widget, or, to follow the usage in MATLAB, a user interface control, is an object in a graphical user interface allowing a user to manipulate input or to produce output. Push buttons are one example of a type of control. Figure 1 shows an example of a GUI with its controls numbered.

CHAPTER 2
AN EXAMPLE OF A GUI

Consider a GUI called G in Figure 1. G is designed to allow the user to create and check pairs of information taking the form (text, color). In addition, the user has the option of sending acceptable pairs, one pair at a time, to another piece of software.

The Controls of the GUI G

The GUI G will be treated as having six controls: a text box, three push buttons,[2] a check box, and a list box.

Control number one in Figure 1 is a text box. It allows the user to enter a string of characters.

Controls two, three, and six are push buttons. Pressing these buttons causes G to perform some procedure such as sending data to another piece of software or closing the GUI. Controls two and three will be referred to respectively as the CHECK button and SEND button.

Control four is a check box. Checking and unchecking the box causes G to toggle between allowing and not allowing the export of data. This control will be referred to as the EXPORT check box.

Control number five is a list box. This list box allows the user to select a color from the predefined set of color choices available in the list.

An Example of a Sequence of User Interactions

With a GUI, even one as simple as G, there are many possible sequences of user interactions. In this example, one such sequence is described. This sequence will be referenced to provide examples throughout this text.

When G is first opened, all of the controls are turned ON except the CHECK and SEND buttons. Controls which are turned OFF will appear darkened as controls 2 and 3 do in Figure 1.

[2]One of the push buttons is the ×-button appearing in the title bar of G. In any GUI, the ×-button has one job: to expedite the closing of the GUI. The other buttons in the title bar will be ignored throughout this text. As this discussion progresses it will be clear that these other title bar buttons do not play a role in the behavior being modeled.

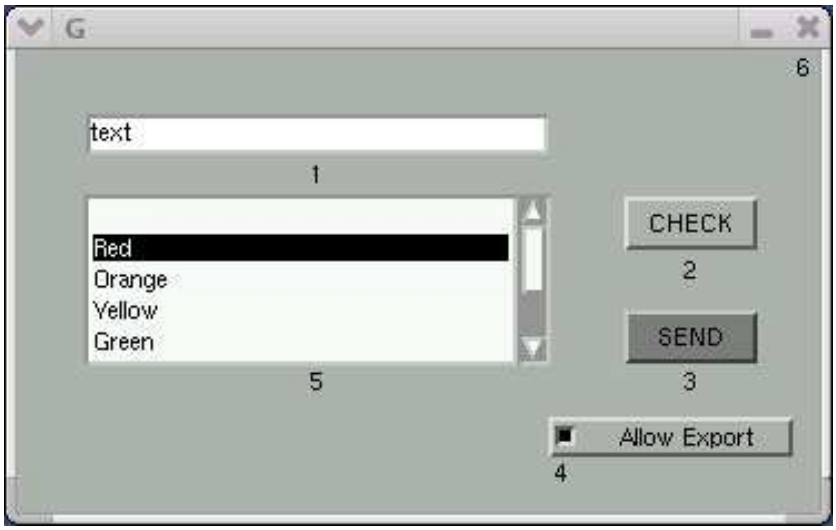

Figure 2: The GUI G with CHECK Button ON.

When the user has entered text and has selected a color, the CHECK button is turned ON as seen in Figure 2.

At this point, the user could enter different text, select a different color, or press the CHECK button. If the user presses the CHECK button, the text/color pair will be tested to see if it is acceptable.[3] If the pair is acceptable and the EXPORT check box is checked, the CHECK button will be turned OFF and the SEND button will be turned ON. Figure 3 displays this condition.

To complete this sequence, the user can press the SEND button. This button press sends the acceptable pair to some other piece of software and returns G to its initial condition. The initial condition of G is shown in Figure 1.

Do note that had the text/color pair been unacceptable when the CHECK button was pressed, the text would have been cleared from the text box, the color would have been unselected, and the ON/OFF states would have been returned to their initial settings. G would then appear as it did when it was first opened.

[3] Acceptability is determined by predefined rules in the GUI's code.

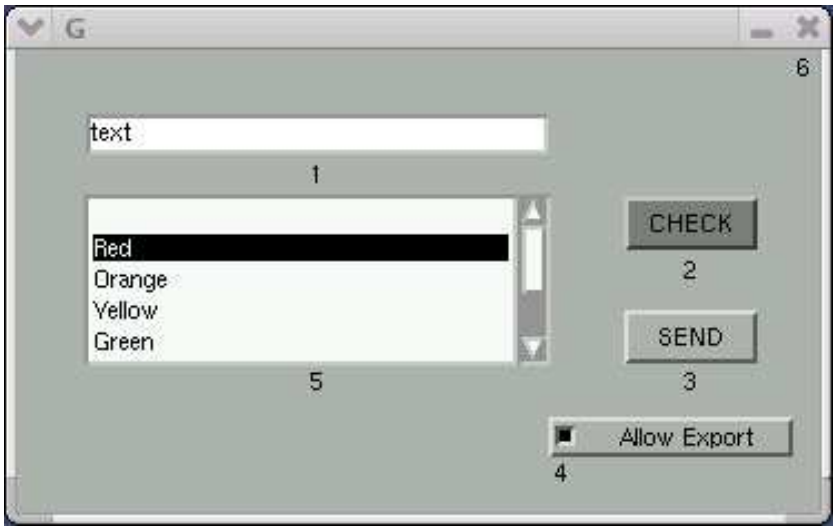

Figure 3: The GUI G with SEND Button ON.

CHAPTER 3
REPRESENTATIONS OF GUI ARCHITECTURE

Event-Based Time

Events

An *event* is defined to be any instance of the provocation of a *procedure* through user interaction with a control. A procedure is the body of functions, routines, and commands performed in response to user interaction with a control.[4] Each user interaction in the sequence in Chapter 2 is an example of an event.

Time

Time in a GUI will not be measured in seconds or minutes. Instead, time will be measured in events and proceed from one.

Time one is when a GUI is first opened. It is time two after the first event occurs, and so on. With this interpretation of time, the i^{th} user interaction moves time from time i to time $i + 1$.

Condition of a GUI

The *condition* of a GUI at a time i is the user data contained in the GUI and the GUI's access-state.[5] The condition can be represented as a vector, c_i, with components d_i and a_i. That is,

$$c_i = \begin{pmatrix} d_i \\ a_i \end{pmatrix}. \tag{1}$$

After time one, the condition of GUI is the result of user interaction provoking the execution of a procedure.

GUI User Data

The GUI user data is the information selected or entered by a user. Many controls will be treated as being able to hold data. For example, consider the text box in G.

[4] Later, procedures will be characterized as functions whose behavior depends on the condition of a GUI and user interaction.

[5] These components of the condition will be defined in subsequent sections on pages 7 and 10.

It can hold the entered text. In addition to the controls, it will be understood that a GUI is able to hold information outside of its controls.

The user data contained in a GUI will be represented as a vector. For any GUI with n controls, the user data contained in the GUI at time i will be represented as:

$$d_i = \begin{pmatrix} d_i^1 \\ \vdots \\ d_i^{n+1} \end{pmatrix}, \tag{2}$$

where, for $k < n + 1$, d_i^k is the user data contained in the GUI's k^{th} control. The $(n+1)^{st}$ component of d_i, d_i^{n+1}, is the user data contained in the GUI and not in any of the controls.

Sets of Possible User Data

The set of possible user data held in the k^{th} control depends on the type of control. Let r_k represent the type of the k^{th} control and let $sd(r_k)$ represent the set of possible user data to be held by a control of type r_k. Then, each d_i^k, for $k < n + 1$, is an element of its corresponding $sd(r_k)$.

Consider five types of controls for examples of $sd(r_k)$. The following will illustrate the sets of possible user data for push buttons, check boxes, text boxes, popup menus, and list boxes.

A push button, like G's CHECK button, can be pushed to provoke some behavior. Since push buttons do not serve to enter data, but to provoke simply some procedure, a push button will be considered to hold no data. That is,

$$sd(push\ button) = \{\epsilon\}, \tag{3}$$

where ϵ refers to an empty, or null, string, as it is used in automata theory.[6]

A check box is used to allow the user to set a particular characteristic of behavior. In G, the check box toggles between allowing the export of a pair or not. The data that can be held in a check box then is of the form true or false, 1 or 0, in reference to the characteristic of behavior, so that

$$sd(check\ box) = \{1, 0\}. \tag{4}$$

[6]This use of ϵ can be found in [$HUl1969$].

A text box allows the user to enter a string of characters. Recall that G's first control is a text box. While specific GUIs, like G, can be designed to have particular strings be unacceptable, in general, there is no limitation on the entries. Thus,

$$sd(\textit{text box}) = \{\textit{all possible strings}\}. \tag{5}$$

An example of a popup, or drop-down, menu is what appears in response to clicking on the commonly found "File" item in many GUI menus. Another example of a popup menu can be found in many electronic forms where the user is asked to select a country of residence. In popup menus, a list of choices is made available, and the user is allowed to select one of the choices. The set of possible user data held in a popup menu is then the set of choices in the menu.

$$sd(\textit{popup menu}) = \{\textit{choices in menu}\} \tag{6}$$

The fifth control in G, which displays a list of colors, is an example of a list box. List boxes are similar to popup menus in that they offer the user a list of choices. List boxes are distinct in that they may allow the user to select more than one of the choices. That is, in general, the user may select any subset of the set of choices. To denote this, the power set of the choices, $\mathcal{P}$ is used.

$$sd(\textit{list box}) = \mathcal{P}(\textit{choices in list}) \tag{7}$$

Clearly, this is not an exhaustive treatment of all of the possible control types. It is possible, however, to describe the set of possible user data for any user interface control.

Examples of User Data Vectors

When the example of a GUI, G, is first opened, the EXPORT check box is checked, and there is no other data being held. Referring to Figure 1, the EXPORT check box is the fourth control. Then, the user data vector at time one is:

$$d_1 = \begin{pmatrix} \epsilon \\ \epsilon \\ \epsilon \\ 1 \\ \{\epsilon\} \\ \epsilon \\ \epsilon \end{pmatrix}, \tag{8}$$

where, again, ϵ refers to the null string. The second, third and sixth components of d_i will always be ϵ for G, since they are push buttons.

Later, after the text was entered and a color was selected, the user data held in the GUI became

$$d_3 = \begin{pmatrix} text \\ \epsilon \\ \epsilon \\ 1 \\ \{Red\} \\ \epsilon \\ \epsilon \end{pmatrix}. \tag{9}$$

Hence, there is text in the text box, the first control, and Red has been selected in the list box, the fifth control.

If G had been designed to hold and send multiple pairs, the seventh component of d_i would be a data structure holding previously selected pairs.

Access-States

Each of a GUI's controls can be ON or OFF. For a control to be ON implies that a user has access to the control. A control which is OFF is inaccessible to the user. This simple state, as it might be called in [*Alh*2003], will be referred to as the *access-state* of the control. The collection of the ON/OFF states of each of the GUI's controls at a particular time will be referred to as the GUI's access-state.

Access Vectors

Using a common convention, ON is represented as a 1, while OFF is represented as a 0.[7] It is natural to represent the access-state of a GUI with a vector of 0's or 1's capturing the ON/OFF state of each control.

This access-state vector, or simply *access vector*, will appear as follows, with the superscript and the subscript of each component denoting the index of the control and the time respectively.

[7] A quick overview of the use of this convention in electronics appears on page 74 of [*Bec*1981]. Two distinct states for electric leads are distinguished. A lead can either carry a signal or not. These states are designated as 1 and 0 respectively.

$$a_i = \begin{pmatrix} a_i^1 \\ \vdots \\ a_i^n \end{pmatrix}. \tag{10}$$

That is, a_i is the vector of access-states of the controls at time i, so that a_i^k is the access-state, or ON/OFF state, of the k^{th} control.

A GUI with n controls has at most 2^n possible access-state vectors. This computation is simple given that there are n components of a_i with 2 choices for each component.

It is likely that this number of possible access vectors will be reduced as information regarding the behavior of a particular GUI is taken into account. For example, a control may be designed to be ON at all times. Then, the largest number of possible GUI access-states for such a GUI with n controls is 2^{n-1}. This pattern continues with additional instances of controls with fixed access-states.

If the access vector is a vector of zeros, then the GUI is said to be *closed.*[8] An *open* GUI is not closed.

Examples of Access Vectors

When G first opens, all of the controls are turned ON except the CHECK and SEND buttons. These buttons are the second and third controls, so that the access vector at time one is:

$$a_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \end{pmatrix}. \tag{11}$$

At time three, after the text has been entered and a color has been selected, the CHECK button is turned ON, so that

[8] An active or open GUI has at most $2^n - 1$ distinct access-states. The minus one accounts for the fact that a vector of all zeros denotes a closed GUI.

$$a_3 = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 1 \\ 1 \end{pmatrix}. \tag{12}$$

Comparing a_1 and a_3, it is easy to see that the only difference is in the second component. The second component refers to the second control, the CHECK button. Recall that this button is OFF, 0, at time 1 and then turned ON, 1, at time 3 in the example sequence.

User Interactions

For any GUI with n controls, the user interaction at time i will also be represented as a vector. That is,

$$u_i = \begin{pmatrix} u_i^1 \\ \vdots \\ u_i^n \end{pmatrix}, \tag{13}$$

where u_i^k is the user's interaction with the GUI's k^{th} control.

Sets of Possible User Interactions

As with user data, the possible user interaction with a particular control depends on the type of the control. A push button only allows the user to push it or not, while an editable text box allows the user to enter text or not. For each type of control, there is a set of possible user interactions. Let r_k represent the type of the k^{th} control and $su(r_k)$ represent the set of possible user interactions for that type. Then, each u_i^k is an element of $su(r_k)$.

The possible user interactions with any type of control, denoted here as $su(r_k)$, will be characterized as a set with two elements. The two elements represent the control being used or not used. In some cases, the element representing use of the control will be a variable representing the user's input or selection.

As with the user data vectors, five control types will serve as examples. The sets of possible user interactions for push buttons, check boxes, text boxes, popup menus, and list boxes will be presented.

Consider push buttons as the first example. A push button can be pushed or not. Its corresponding set of possible user interactions is simply $\{1, 0\}$, where one represents the push of the button and zero represents the button being not pushed. That is,

$$su(push\ button) = \{1, 0\}. \tag{14}$$

Though the behavior and result of checking and unchecking a check box is different from a push button, the set of possible user interactions for a check box is the same as for a push button.

$$su(check\ box) = \{1, 0\}. \tag{15}$$

The 1 in Equation 15 implies the check box has been used. With each use, the check box toggles between being checked and unchecked. Said another way, if the user data held in the check box is a 1 and the user interacts with the check box, the user data held in the check box will then be zero, and vice versa.

A text box allows the user to enter text. In general,

$$su(text\ box) = \{t, 0\},\ t \in sd(text\ box), \tag{16}$$

where t represents the entered text, and t is, as noted, an element of the set of possible user data for a text box.

In a popup menu, the user may select one of the choices available in the menu. Using v to represent the selection,

$$su(popup\ menu) = \{v, 0\},\ v \in sd(popup\ menu). \tag{17}$$

A list box displays a set of choices as a list. With a list box, the user may select one or more of the choices. To characterize the set of possible user interactions with a list box, let V represent the user's selection. Then, with zero representing the list box not being used,

$$su(list\ box) = \{V, 0\},\ V \in sd(list\ box). \tag{18}$$

Similar characterization may be done with other types of controls. Following the convention used here, zero will be used to represent the case when a control is not used. Then, an appropriate variable can be chosen to represent the control's use.

Examples of User Interaction Vectors

The first user interaction in the sequence in Chapter 2 is the entry of text. The user interaction at time 1 is then:

$$u_1 = \begin{pmatrix} text \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}. \tag{19}$$

The CHECK button is the second control in G and it is pressed at time 3 in the example sequence. The user interaction vector at time 3 is

$$u_3 = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}. \tag{20}$$

Procedures as Transformations

Earlier in the text, procedures were introduced as the body of functions, routines, and commands performed in response to user interaction with a control. A procedure can also be construed as a function whose output is the condition of the GUI. With each user interaction, a procedure is called which may result in new user data and new access-states. That is, the procedures result in a new condition vector.

Construed as a function, procedures will be referred to as transformations. Transformations depend on user interaction, user data held in the GUI, and the access-states of the controls. That is,

$$\begin{aligned} T &: \mathbb{U} \times \mathbb{S} \to \mathbb{S} \\ c_{i+1} &= T(u_i, c_i), \end{aligned} \tag{21}$$

where T represents the procedure as a transformation, $\mathbb{U}$ is the space of u_i's, and $\mathbb{S}$ is the space of c_i's. As an argument of T, c_i captures the dependence of T on both d_i and a_i.

Sequences of Transformations

A sequence of user interactions results in a sequence of procedures, and a sequence of procedures can now be described as a sequence of transformations. As there are many possible sequences of user interactions, there are many distinct transformation sequences possible in a GUI. A GUI's behavior will be considered to be the set of all possible transformation sequences. Call this set Γ.

A transformation sequence has the condition of the GUI as its input and output. That is, the user data and access-states undergo transformations through the sequence of procedures which are provoked by the user interactions.

A nonempty transformation sequence will be said to be *complete* when its final procedure closes the GUI, or when the GUI is returned to its initial condition, c_1.

Subsets of the Set of Transformation Sequences

Let m be the number of events in a complete transformation sequence. With time beginning at one, a transformation sequence with m events completes at time m. Thus, at time $m + 1$ the GUI is either closed or set to have another transformation begin. For each conceivable time, there is a corresponding set of transformation sequences with m events. Then for any m, $\mathbb{T}_m$ will be used to represent this subset of Γ.[9]

The example sequence in Chapter 2 is therefore a member of $\mathbb{T}_4$. The example sequence had four user interactions: text entry, color selection, CHECK press, and SEND press. Following the press of the SEND button, G was returned to its initial condition.

Modeling the Example of a Sequence

Now, with the representations defined, the sequence example in Chapter 2 can be rearticulated in the language of the model. This will be done with brief descriptions of the sequence for reference along the way.

G is first opened. The user data and access vector at time 1 are:

[9]The nonempty $\mathbb{T}_m$'s partition Γ. That is, as described in [*Hal*1960], if X takes the role of Γ, these $\mathbb{T}_m$'s form "a disjoint collection C of nonempty subsets of X whose union is X." Clearly, if $\mu \neq \eta$, a transformation sequence belonging to $\mathbb{T}_\mu$ cannot also belong to $\mathbb{T}_\eta$. Hence, the T_m's are disjoint. Then, taking all of the nonempty $\mathbb{T}_m$'s for each viable m in the context of the GUI associated with Γ, it must be that every T in Γ is included in the union. Each transformation has exactly one number, m, of user interactions.

$$d_1 = \begin{pmatrix} \epsilon \\ \epsilon \\ \epsilon \\ 1 \\ \{\epsilon\} \\ \epsilon \\ \epsilon \end{pmatrix}, \quad a_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \end{pmatrix}. \tag{22}$$

The first event was provoked by the entering of *text* in the text box, so that

$$u_1 = \begin{pmatrix} text \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}. \tag{23}$$

It is now time two. The condition of G at time 2 is captured as:

$$d_2 = \begin{pmatrix} text \\ \epsilon \\ \epsilon \\ 1 \\ \{\epsilon\} \\ \epsilon \\ \epsilon \end{pmatrix}, \quad a_2 = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \end{pmatrix}. \tag{24}$$

Notice that *text* is in d_2, and $a_2 = a_1$. The access-states did not change in response to u_1.

The second user interaction is the selection of a color in the list box.

$$u_2 = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \{Red\} \\ 0 \end{pmatrix} \tag{25}$$

Now, G would appear to the user as in Figure 2, and the condition at time 3 is composed of:

$$d_3 = \begin{pmatrix} text \\ \epsilon \\ \epsilon \\ 1 \\ \{Red\} \\ \epsilon \\ \epsilon \end{pmatrix}, \quad a_3 = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 1 \\ 1 \end{pmatrix}. \tag{26}$$

$\{Red\}$ is now d_3^5, and the CHECK button is ON reflected in a_3^1. Next, the CHECK button is pressed.

$$u_3 = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \tag{27}$$

The previous user interaction causes the SEND button to be turned ON and G to appear as it does in Figure 3. The condition at time 4 is as follows:

$$d_4 = \begin{pmatrix} t \\ \epsilon \\ \epsilon \\ 1 \\ \{Red\} \\ \epsilon \\ \epsilon \end{pmatrix}, \quad a_4 = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}. \tag{28}$$

To complete the sequence, the SEND button is pressed.

$$u_4 = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \tag{29}$$

G was designed so that following this sequence, G would be returned to its initial condition. That is, $c_5 = c_1$.

CHAPTER 4
USING THE MODEL IN GUI DEVELOPMENT

As in the last section of Chapter 3, each sequence allowed in a GUI can be modeled using the representations. The usefulness of the model is in developing GUIs that behave as expected. Two ways of achieving this will be explored. First, the model will be considered as a means to test GUIs. Then, it will be considered as a means to develop GUI code.

An Automated Method for Reading GUI Code

With an automated method for reading GUI code and modeling each possible transformation, a developer could discover sequences which allow unwanted behavior[10]. With an entire offending sequence captured in the model, it will be an easier task to correct the GUI's code.

Behavioral rules would need to be known and translated into the model's language. A useful automated method will not only chart out the possible transformation sequences in a GUI, it will be designed to look for sequences which violate behavioral rules. Using this method eases the development of a GUI since it automates the process of discovering rule violations and tracking the user interactions producing such violations.

Implementation of this automated method in GUI development is likely to be an iterative process. A sequence for one iteration is:

1. Distinguish Behavioral Rules
2. Put Behavioral Rules in Terms of Model
3. Use Automated Method to Discover Sequences which Produce Violations
4. Implement Corrections in GUI Code
5. Rerun Automated Method

[10] An example of unwanted behavior is having particular controls that are ON at the same time and should not be. Another example of unwanted behavior is having a particular sequence of user interactions occur that should not be allowed. As will be seen, this sort of offensive behavior can easily be captured with the representations presented.

An Example of Implementing the Method

For an example of this process, consider the development of the example GUI G. The first three steps of one iteration will be followed. This will suffice to demonstrate the viability of this method.

First, the following are some of the behavioral rules of G:

(1) If the EXPORT check box is not checked, then the SEND button should not be ON.

(2) Any CHECK of acceptable pairs, when EXPORT is unchecked, should reset G to its initial condition.

(3) A user should not be able to enter text or make a color choice and then SEND without pressing CHECK first.

(4) The CHECK and SEND buttons should not both be ON at the same time.

Second, each of the above rules can be formulated in the language of the model. Recall that d_i, a_i, and u_i refer respectively to the user data, access states, and user interaction at time i. The superscripts refer to the index of the control.

(1) If $d_i^4 = 0$, then $a_i^3 \neq 1$.

(2) If $d_i^4 = 0$ and $u_i^2 = 1$, then $c_{i+1} = c_1$. Recall that $c_{i+1} = c_1$ means that $d_{i+1} = d_1$ and $a_{i+1} = a_1$.

(3) If $u_i^1 = t$ or $u_i^5 = V$, then $u_{i+w}^3 = 1$ iff $u_{i+m}^2 = 1$ for $m < w$ and $u_{i+h}^5 = u_{i+h}^1 = 0$ for $m < h < w$.[11]

(4) For any i, $\neg(a_i^2 = 1 \text{ and } a_i^3 = 1)$.[12]

The automated method can now be used to look for violations of any of the above rules. Not only will the automated method check for violations, it will capture the entire transformation sequence producing the violation.

While G was being developed, this third step was done by hand, without any automated method. Sequences producing rule violations were discovered.

The following discussion charts out an example of a transformation sequence violating Rule 1. This example follows the previous example sequence up until

[11]The variables t and V represent entered text and color choice respectively. These variables are first used in the section on User Interactions beginning on page 12.

[12]"¬" denotes "not".

time 4.[13] That sequence is as follows: *text* is entered at time 1 in the text box; {*Red*} is selected at time 2 in the list box; and the CHECK button is pressed at time 3. So that at time 4:

$$d_4 = \begin{pmatrix} text \\ \epsilon \\ \epsilon \\ 1 \\ \{Red\} \\ \epsilon \\ \epsilon \end{pmatrix}, \quad a_4 = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}. \tag{30}$$

Then, instead of pressing the SEND button, as was done in the example sequence, the EXPORT check box is unchecked. That is,

$$u_4 = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}. \tag{31}$$

This results in:

$$d_5 = \begin{pmatrix} text \\ \epsilon \\ \epsilon \\ 0 \\ \{Red\} \\ \epsilon \\ \epsilon \end{pmatrix}, \quad a_5 = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}. \tag{32}$$

Notice that $d_5^4 = 0$ and $a_5^3 = 1$, so that Rule 1 has been violated.

Then, the developer can pinpoint the logical failure of the code,[14] make corrections to the GUI code, and rerun the automated method to double-check the corrections.

[13] Refer to the section beginning on page 15 which provides the behavior modeled up to time 4.

[14] Clearly, the behavior in response to checking or unchecking the EXPORT check box needed to be better defined.

A Method for Writing GUI Code

In addition to having an automated method for testing GUIs, it would be useful to have a method for writing GUI code. Such a method may be possible given the accomplishments attained in the research and development of *specification languages*. Wikipedia, [*Wik*2004], offers this definition:

> A specification language is a formal language used in computer science. Unlike programming languages, which are directly executable formal languages used to implement a system, specification languages are used during system analysis and design.

Recall that once behavioral rules for a GUI have been distinguished, they can be rearticulated in the language of the model. The rules, as expressed by the model, are first-order logical statements built on the foundation of set theory. As such, the language of the model offers a specification language for the GUI code.

The development of a method for writing GUI code from this foundation follows work already done in the study of *formal methods*,[15] specifically advancements made with the *B-Method*.

The B-Method

In an overview of the B-Method, the United Nations University International Institute for Software Technology, [*UNU*2004], offers this description:

> The B method essentially deals with the central aspects of the software life cycle, namely: the technical specification, the design by successive refinement steps, the layered architecture, and the executable code generation. Each of the previous items is envisaged as an activity that involves writing mathematical proofs in order to justify its results. It is, the collection of such proofs that makes one convinced that the software system in question is indeed correct.

More specifically, the B-Method uses a mathematically based language, called abstract machine notation, to describe systems. The descriptions come in the form of

[15] The Free On-Line Dictionary of Computing, FOLDOC, describes formal methods as, "Mathematically based techniques for the specification, development and verification of software and hardware systems" [*Fol*2004].

compositions of abstract machines.[16] Through a process of implementation and refinement, these descriptions are turned into "highly maintainable, separately compilable source code (C) and executables" [*BCo*2002].

The B-Method offers a sound example of a method for translating a specification language into a programming language. Following the development and implementation of this example provides an avenue for developing a method for translating a GUI's behavioral rules into executable GUI code.

[16]For more on abstract machines in the context of the B-Method, refer to Scheider's text, *The B-Method: An Introduction* [*Sch*2002].

CHAPTER 5
CONCLUSIONS

Usefulness of Representations

The representations suggested in this text provide a means to model the structure and behavior of GUIs under different sequences of user interactions. The representations provided here can be used to ease the process of developing and testing graphical user interfaces.

Suggestions for Future Development

First, what is wanted is an automated method for reading GUI code and modeling possible transformations. Having this method frees the developer from tracking by hand the minutiae involved in actually interacting with the modeled object, in this case a GUI. With more complex GUIs, the task of modeling the transformation sequences by hand will be cumbersome. An automated method will dispose of these difficulties readily.

Next, it is suggested that a method for writing GUI code from the behavioral rules be developed. This method would begin with using the language of the model to articulate the behavioral rules. The accomplishments in the research and development of the B-Method can be implemented to develop such a method.

Completion

Ultimately, it will be useful to integrate this work into the already existing branches of research. As was seen in the introduction, there is a variety of work being done to examine, improve and expand the usefulness of graphical user interfaces. By providing abstract and generalized representations which form a useful model, the work in this text contributes to a thorough examination of the architecture of particular graphical user interfaces.

LIST OF REFERENCES

LIST OF REFERENCES

[*Alh*2003] Alhir, S., *Learning UML*, O'Reilly & Associates, Inc., 2003.

[*BCo*2002] *B-Core, The B-Technology Company: About B: Method* (8 Feb 2002), http://www.b-core.com/aboutb_method.html, (23 June 2004).

[*Bec*1981] Beckman, F., *Mathematical Foundations for Programming*, Addison- Wesley, 1981.

[*Cal*1994] Callahan, G., *Semiotics and GUI Design* (1994), http://www.stgtech.com/staff/gcallah/tech_articles/semiotics.html, (14 Jan 2004).

[*Fol*2004] *Formal Methods from FOLDOC* (1996), http://foldoc.doc.ic.ac.uk... .../foldoc/foldoc.cgi?formal+methods, (31 May 2004).

[*Hal*1960] Halmos, P., *Naive Set Theory*, Princeton, NJ: D. Van Nostrand Co., Inc., 1960.

[*Hob*1995] Hobart, J., *Principles of Good GUI Design*, UNIX REVIEW, **10** (1995), 37-46.

[*HUl*1969] Hopcroft, J. and J. Ullman *Formal Languages and Their Relation to Automata*, Addison-Wesley, 1969.

[*Kol*2003] Kollar L., *GUIML, creating dynamic GUI objects using XML*, MATHEMATICAL AND COMPUTER MODELLING, **38:7-9** (2003) 893-901.

[*Mat*2004] *MATLAB Help Documentation*, http://www.mathworks.com... .../access/helpdesk/help/techdoc/creating_guis/creating_guis.shmtl, (16 Mar 2004).

[*Sch*2002] Schneider, S., *The B-Method: An Introduction*, Palgrave, 2002.

[*UNU*2004] *The B-Method Course*, http://www.iist.unu.edu/home... .../Unuiist/newrh/II/2/1/6/page.html, (24 June 2004).

[*VKa*2003] Virvou M. and K. Kabassi, *Experimental studies within the software engineering process for intelligent assistance in a GUI*, JOURNAL OF UNIVERSAL COMPUTER SCIENCE, **9:1** (2003) 51-85.

[*VKa*2004] Virvou M. and K. Kabassi, *Evaluating an intelligent graphical user interface by comparison with human experts*, KNOWLEDGE-BASED SYSTEMS, **17:1** (2004) 31-37.

[*Wik*2004] *Specification Language: from Wikipedia the Free Encyclopedia* (2004), http://en.wikipedia.org/wiki/Specification_language, (31 May 2004).

VITA

Eric Freeman Mann was born on November 18, 1974 in Grand Rapids, Michigan. He received all of his undergraduate education in Grand Rapids. He was a student at Sherwood Park Elementary School, and then he attended City High-Middle School, graduating in 1992. He received his Bachelor of Science Degree in Mathematics from Aquinas College in May 1997 graduating Cum Laude. In the 1996-1997 academic year, Eric took mathematics courses at Grand Valley State University.

Eric was a faculty member at the Metropolitan Arts Institute in Phoenix, Arizona from 1998 through 2002. There he taught high school math courses and served on Board of Directors.

Eric began his graduate education at the University of Tennessee Space Institute in 2002. While he was at Tennessee, he worked as a graduate research assistant in the 2002-2003 and 2003-2004 school years, and he served as the Institute's Student Government President in the 2003-2004 school year. He received his Master of Science Degree in Mathematics from the University of Tennessee, Knoxville in August 2004.