Development of the Computer Assisted Circuit Editing System (CACES)

Technical Report 94-19

David B. Lambert

Project Proposal
for
Masters of Computer Science and Engineering
Department of Computer Science and Engineering
Auburn University, Alabama

Submitted to:

Dr. Loretta Moore, Advisor

Dr. A. Scottedward Hodel

Dr. W. H. Carlisle

December 2, 1994

Introduction

Since the introduction of computers that were inexpensive enough to be available in the schools, there has been a demand to use these machines in education. Initial attempts at building computer based learning systems were limited by CPU's that did not have the processing power to run the needed software and the lack of development tools to help build good systems. The initial computers usually had text-only displays and limited storage. These displays limited the software designer as to what types of data (i.e. alphanumeric only) could be presented to the student, and the storage limited the amount of information that could be presented.

The text-only interface of these early systems often made for software that was not very intuitive to use. In some cases, there was more instruction needed for the user to use the tutoring system than for the topic the system was trying to teach [1]. This resulted in computers not being used very well in education. Teachers found that it was easier to teach using the older methods.

As time progressed through the 1970's, computers became more powerful while the prices continued to drop. Manufacturers were also learning that a machine without software was not useful and began to bundle them with software. For example, one of the first products that manufacturers provided with every computer (besides some sort of an operating system) was a BASIC interpreter. BASIC was a simple, easy to learn programming language that opened a new door to educators. BASIC was a language that allowed educators to teach basic programming principles [1] to their students. This began the process of integrating computers into everyday use in education.

Though this brought about some improvements, educators were still stuck with software that was either written by someone else for another class or one that lacked many of the features that would make them easy to use. Though there was some progress, the early software was still not very intuitive and was often very difficult to modify. The software that was modifiable still required programming skills that are beyond the average teacher and ultimately required more time to set up a course on the computer than using the traditional methods [1]. Teachers needed tools that would lower the preparation time to create a lesson, but still be complete and easy to use. If sufficient tools existed, they would aid in making a lesson easier to create and would facilitate the use of computers in education.

The goal of this project is to produce a graphic design tool to aid in creating drawings of simple electrical circuits for the Electrical Engineering course on Controls. This tool is being developed in conjunction with a larger Computer Aided Instruction (CAI) system for the department. This tool is to help make the development of portions of problems in the CAI system easier for teachers. The tool requires no programming knowledge, only knowledge of electrical circuits. This is done by patterning the system after drawing packages that many people already know how to use. The user is presented with a palate of circuit elements to build the circuits.

This addition to a computer aided instruction system will ultimately make the use of such a system easier. This will allow more people to begin to use computers in education and that will ultimately help the students learn better about a wide variety of topics.

Related Research

There are other programs that have been written to help tutor students in Electrical Engineering courses. Several of these have some limitations that will be explained and discussed below. The Computer Assisted Circuit Editing System (CACES) will attempt to either remove these limitation or lessen their importance. This will be discussed below.

Circuit Exerciser

The first such example is the Circuit Exerciser written by Yoshikawa, Shintani, and Ohba [2]. This system was designed to help students to learn more about electric circuits by providing the students with drill problems and then helping the student solve them.

Circuit Exerciser randomly generates an electrical circuit and displays it for the student. The system then picks a problem dealing with the circuit and begins to ask the student questions about the circuit. The student enters a response to the question and it is analyzed. While the student is trying to solve the problem, the system is solving the problem for comparison with the students response. If the student's response is not correct, the system uses an artificial intelligence technique to try and deduce where the students made a mistake.

The system's graphics ability is used to display a representation of the circuit to the student. The method of input/output for the student is via text. The student is shown a picture of a circuit and is then asked to solve a problem and enter the answer. Since the problems are generated randomly (by the system), the teacher has little control over the problem. Randomly generated circuits can create problems that are very difficult to solve.

CACES will be used by our tutoring system to allow the teacher to build a database of circuits that will be usable by the system. Allowing the teacher to create the circuits will eliminate the types of problems that would be to difficult to solve by undergraduate students while allowing the teacher to create difficult problems that could be given to the more advanced students. CACES could also be used to allow the student to enter a drawing of a circuit instead of just entering a textual answer. This will expand the types of questions that the student can be asked.

The Circuit Exerciser also has one other flaw that the authors discuss. Since this system's only method of input is via a keyboard, there is not an easy way for a student to enter a mathematical equation. The two options here are to either develop a "language" that will let the students represent equations using standard text or build an equations editor that will allow the user to create them graphically. The former method means that the students need to spend time learning the language instead of time learning about circuits. The latter method can drastically reduce this time. The tutoring system that we are proposing will include an equation editor for use by the student and teacher.

KIRCHHOFF

One step closer to the computer aided instruction system that is being developed at Auburn is the one developed by de Coulon, Forte, and Rivera called KIRCHHOFF [3]. This system is designed to help teach a first course in electrical circuits. It requires no programming knowledge and no lengthy training. The system has the capability to help teach students with no knowledge of electrical circuits.

KIRCHHOFF provides the user with a graphical representation of a circuit in the upper portion of the screen while the lower portion has a list of actions that the user may perform on the circuit. These options are used to guide the user into a successful solution of the problem.

One interesting point to note about KIRCHHOFF is that it tries to minimize guidance and feedback to the user [3]. The user may start any of the actions that are listed on the lower portion of the screen, even if they are incorrect. The user will be given an option that will allow the user to see hints, one at a time, and to see the errors that have been committed.

The ultimate goal of this system is to have the user write the nodal equations about the given circuit. KIRCHHOFF does this by having the user label all current sources and then write the equations summing the currents into and out of a node. KIRCHHOFF handles the input of mathematical equations by writing the general form of the equations and then having the student modify some of the values in the equation.

This system offers several items that are desirable in the tutoring system that CACES will ultimately be a part of. The first of these traits is that it allows the teacher to build a library of circuits to be used in the system. This leaves the teacher a level of control as to exactly how difficult a problem will be presented to the student. Though this may require a large amount of time initially to build a large enough library to use the system, the level of control it offers is advantageous.

KIRCHHOFF also has a built in equations editor. This feature allows the student to enter equations in a form they would normally use on paper and reduces the training time in learning to use the tutoring system. The only other way to enter a complex equation would be to have some sort of language the student would have to learn in order to input equations.

The only major drawback to the KIRCHHOFF system is that it is somewhat limited in what it is able to teach. As its name implies, this system is designed to teach students how to write Kirchhoff equations about simple electronic circuits. Though it does this job well, it is not well suited to teach about other aspects of circuits.

COPERITE

Another example of computerized instruction is the Computer-aided tool for Power Engineering Research, Instruction, Training and Education (COPERITE) written by Chowdhury and Clark [4]. Though this system goes a bit beyond just a computer aided instruction system, it does provide some of the items that are absent in Circuit Exerciser. COPERITE attempts to teach users about the building power systems on the computer and then see the results.

Though this may not seem at first glance to be an instructional tool, it does offer the user a chance to build systems and learn from their mistakes without building the real thing. The COPERITE system teaches the user by providing immediate feedback about possible mistakes while the user is still drawing the circuit. As each item is placed on the drawing area, a rule is created in a PROLOG database that represents the power system. Also contained in the database are rules to detect states that would cause a critical failure and keep the user from leaving out necessary components. When one of these critical states is reached, the student is informed immediately.

Feedback to the user is given via the graphical user interface. COPERITE has the ability to show the user exactly where an error has occurred. This allows the user to immediately see the error and correct it, which makes the identification of a mistake as early as possible in the design of the power system.

The major drawback to the COPERITE system in regards to it being a tutoring system is that it requires a good deal of subject knowledge before being able to use the system effectively. COPERITE is mainly designed to help instruct users with good working knowledge about power systems [5]. This is not acceptable for users who only want to learn about the subject. For those users, this system would be confusing.

CACES intends to work from the idea that the user will have very limited knowledge of the subject and will help teach the student the fundamentals. Later, by increasing the problem complexity, the system could be used in a similar manner as COPERITE to help those who already know the fundamentals refresh their knowledge. The CACES system will provide the graphical support needed by the tutoring system.

Though the COPERITE system steps closer to what we are trying to build by making use of a graphical user interface to represent a circuit, it still falls short on what is needed to make it a good undergraduate tutoring system.

CAL Programs

Lastly, there are a series of packages that were developed by Cvetkovic, Seebold, Bateson, and Okretic [5]. This represents a series of programs that were developed with the cT(tm) programming environment. Several packages were created using this environment to teach such Electrical Engineering courses as:

- Signals and Systems Tutorials
- Electromagnetic Field Theory
- Power Electronics
- Monte Carlo Simulation Methods
- Selected topics in Telecommunications

Most of the software design for these applications was done by undergraduates as their senior design projects. It was felt that these students were in a better position to help write software for other undergraduates than the teachers of the courses.

The main approach this software uses to teaching is to divide material into units with each unit containing as little information as possible so as not to clutter up the screen. Mathematical formulae are presented first, followed by text which explains the key concepts behind the equations. All the calculations are carried out in real time and the process is then observed as an animated image. This is followed by more detail of the theory.

In the sections of the software where theory is being introduced, the student is presented with a mix of text and graphics to illustrate the subject. When the system needs a response, the student can either type in an answer, or use on-screen tools to modify the graphical image. The software can then update the image and determine the validity of the student's response.

This software appears to be a dramatic step toward the type of tutoring system that CACES will be a part of. This system provides a good blend of text and graphics to help demonstrate complex ideas. The system also provides good user input methods so that the student can visualize the changes being made.

Lastly, this software has been used to build systems that teach a variety of different electrical engineering topics. This shows some of the flexibility of the system. The only drawback that I found with this system is that each “course” the software teaches must be coded by someone with expertise in the area as well as good programming skills. Though this is often not a problem for teachers in the school of Electrical Engineering, this could represent a problem in other areas where computer knowledge is not as great.

Architecture of the System

An integral part of the tutoring system to being developed here at Auburn University is the circuit editor (CACES). CACES will provide the user an easy to use system for creating simple electronic circuits. The major appearance of the system will be that of a drawing package that provides the user with the basic elements needed to build a circuit (resistor, capacitor, voltage source, etc.). The creation of a circuit in CACES will be similar to that of building a real electrical circuit.

When the system is first started, the user is presented with a blank drawing area along with icons to represent the objects to be placed in the circuit. The user will be able to pick up the object and place it in the drawing area. This will be what Polson & Richardson call a first-person or direct manipulation interface [6]. This means that the user has a feeling of working directly with the objects that are represented by the computer. If the user has some knowledge of building circuits, then the time to learn to use this system is reduced.

To further this idea of direct-manipulation, once the user has placed any of the objects that are needed in the circuit on the drawing area, they will need to connect the objects together, just like they would if they were building a real circuit. This will be accomplished by having the user select a "wire" tool and connect the items together. This will be similar to connecting two elements together with real wire.

Direct manipulation helps to avoid a common problem in user interface design called task-mapping [6]. In many user interfaces, there is a large gap between the goals a user has when starting to use a system and the tasks that the user interface provides. For example, a user may want to create a circuit with a voltage source, a capacitor, and a resistor. If the user interface requires the user to create a definition of "voltage source", place it, and lastly make a connection to it, users are going to have more difficulty using the system because these actions are not natural to what they would do in real life. This difference in tasks can be reduced by having the user interface actions more closely mimic the real-world actions that the user is trying to do. In this case, having a "voltage source" icon that the user must simply drag to its location and then draw lines to other items to make connections is a more natural action.

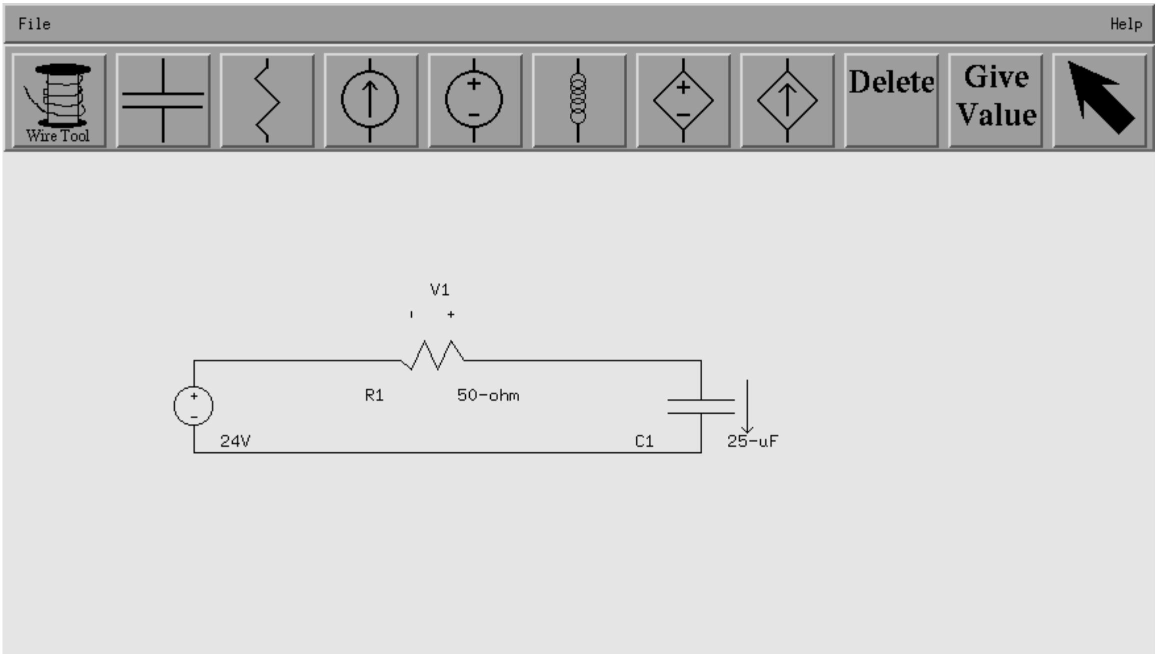

Figure 1 - Main Window of CACES

Each element that the user can place is a generic element that represents a group of similar items. For example, CACES provides a resistor icon, but initially makes no distinction between a 10-ohm resistor and a 100-ohm resistor. In order to make a complete circuit, the user must give each element that is placed on the drawing area some distinct value. This will be done by having the user select the placed object and then give the object a value. This will allow for a full range of different circuits to be drawn and therefore used in the tutoring system.

This process of drawing an electrical circuit ultimately produces a simple bitmap representation of what a circuit looks like. Though this form is useful to humans, it is not in a form that the computer can use to simulate the circuit. The computer needs to know which objects are connected together and needs to know some information about each object. For example, the computer needs to know that a 10 ohm resistor is connected to a 10 volt voltage source. This form is more useful in terms of simulating the circuit.

Using the System

As has already been stated, CACES is modeled after a drawing package. The user in presented with a palate of items that can be used to build a representation of an electrical circuit. In order to place an object in the drawing area, the user will use the mouse to first select the object in the palate (by pressing the left mouse button) and then moving the cursor to the approximate location on the drawing area to place the object. Next, the user can press and hold the left button and the object will appear on the drawing area. As long as the user holds down the left button, the user can move the object around the drawing area to place it in it's exact location. The user can repeat this procedure to place other objects in the drawing area.

Next the user may want to give each of the objects a specific value and a label. This is done by first selecting the "Give Values" tool and then placing the cursor over one of the objects and pressing the left mouse button. The program will display a pop-up window to the user that contains fields to enter data about the object. The popup is shown in figure 2. Data entry is done with the keyboard. This can be repeated for each of the elements in the drawing area.

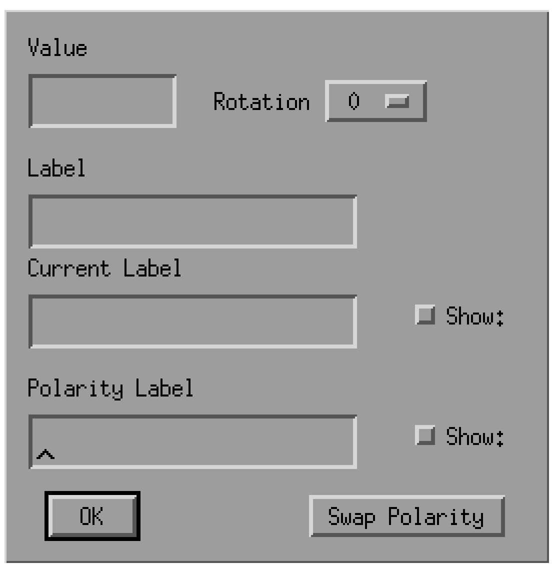

Figure 2 - Give Value Popup

The user will need to connect the items together in order to make a complete circuit. Each element in the circuit editor has two points that can be a connection point. This is similar to the two ends of the real items that these icons are intended to represent. On each of the elements, there are two line segments extending from of the top and bottom of each element. The user can select the Wire Tool and two of these line segments to make a connection.

There is one more feature that CACES offers the user in creating circuit representations. This is the ability to rotate the objects. In the case of an electrical circuit, certain elements (i.e. voltage sources) have an inherent polarity in them. This means that the real-world object makes a distinction between the top connector and the bottom connector. This is like the positive and negative leads of a battery. In order to draw a complete range of circuits, the user may need to swap the polarity of the object on screen. This can be done in one of two ways. The first involves having the user rotate the object until the polarity is correct. The other involves pressing a swap polarity button on the popup.

While this covers the basic functions needed to create a circuit, there is also a means to alter the drawing once it has been created. This is done with two additional tools. There is a moving tool (represented with a large arrow) that the user can use to pick up a placed object and drag it to a new location. There is also a deletion tool that will allow the user to completely remove an object from the drawing area.

When the user has completed a drawing, the system provides a means of saving the drawing to a file. This is done by selecting the Save option from the File menu. When the user does this, a dialog box is presented that lets the user enter a file name. The program then take the drawing and stores it in the requested file. The user can later load this drawing again using the file name and then continue to edit the drawing. The exact format of this file is discussed in appendix A. Ultimately, this file will be used by the tutoring system to model the circuit.

Design Problems

In writing the CACES system, several problems arose as to how to handle the drawing of the objects. These ranged from how to represent the objects to dragging an object across the drawing area without it destroying any other part of the drawing. This section describes the approaches which were taken in solving the these problems.

Object Representation

The first problem to be addressed was internal representation of the objects for computer display. The initial idea was to simply have a fixed bitmap drawing of each object. This would make it easy for the system draw the object. However, It was quickly determined that the objects would need to be rotated for satisfactory drawings. For example, in Figure 3, the left circuit, though technically correct, does not look like the normal form of this same circuit on the right.

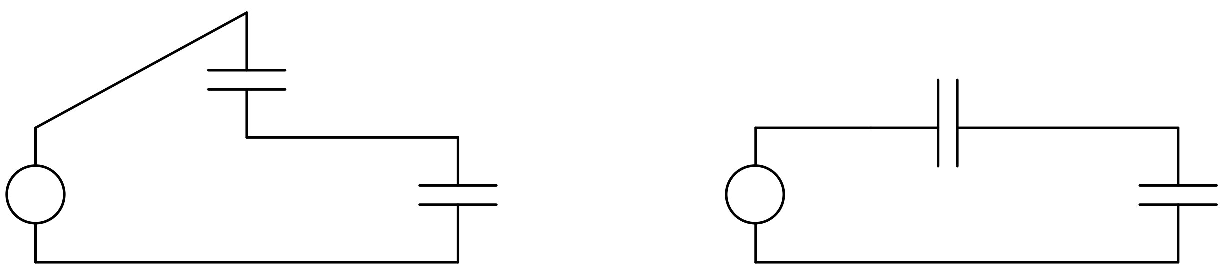

Figure 3

Using a bitmap representation of an element in this case would not be a good choice since rotating a bitmap is not a very easy thing to do. Bitmaps are stored as arrays of values that identify each point in the bitmap as either on or off. In order to rotate one of these bitmaps, a routine would have to be written to examine each point and generate it's equivalent point rotated. With a 64x64 pixel bitmap (the size used here), this would result in examining 4096 pixels each time a bitmap needed to be rotated.

An alternative to bitmapped representation was to represent each object as a series of sub elements. These sub elements were selected to be lines, circles, boxes, and arcs. In doing this, a description of the object to be drawn would be created. The result would be something like:

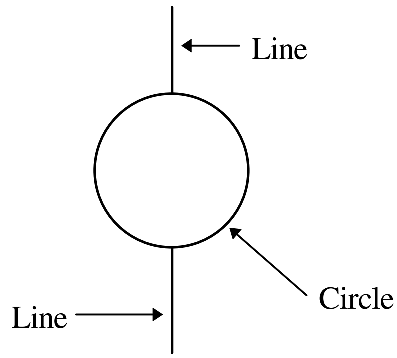

Figure 4

A representation of the object like this drastically reduces the number of points to examine since each of the sub elements needs only two points in order to draw. The example in Figure 4 results in six points that need to be examined and altered in order to rotate the object. These six points would be the end points of the two lines, and the two points that defines the size of the circle.

This lead to generating a definitions file for each of the objects that would be used within CACES. This consists of a series of points and a value to indicate which of the sub elements those points represents. Once such a list of points is created, rotating the object is done by taking each point and calculating its new location rotated about the center of the object.

The way we calculate the point's new location is by using a translation matrix. The translation matrix is built based on the degree of rotation. When that is done, the new point rotated around (0,0) can be calculated as:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

While doing the matrix calculation could be costly in terms of computation, we can simplify this matrix calculation to:

$$x' = x \cdot \cos\theta - y \cdot \sin\theta$$
$$y' = x \cdot \sin\theta + y \cdot \cos\theta$$

Adding the ability to rotate the objects ultimately gave the system more flexibility. A side benefit to rotation was the ability to alter the polarity of some of the objects in the circuit by rotating them 180 degrees. This helps CACES to draw a complete electrical circuit.

Making Connections

Once the objects are placed in the drawing area, the next problem to arise involved connecting the items together. Each object has a set of points associated with it which represents where the user can connect wires. There is a parallel to this in the real world. For example, a resistor has two ends that the user can connect wires to, therefore the resistor object needs two ends that the user can connect our “wire” to. In the case of the circuit elements, this is denoted by a line extending out of the top and the bottom of the object. At the end of these lines is the connection point.

In order for a user to connect two objects together, they must use the wire tool to connect two of the connection points together. This was necessary because the tutoring system will need to know exactly how the objects are connected together. CACES enforces this by only allowing the user to start or end a wire segment on one of the connectors.

Originally, these connection points were assumed to be in the same location for all objects. Though this made the initial development easier, it would have prevented representing certain types of items (i.e. objects with both connectors on the bottom). Since the decision was made to go ahead and move the object definition out into a file, it was decided to also move the locations of the connections there also. The result would be a system that can represent a wider variety of items.

Moving Objects

In the initial stages of building this system, the problem of how to exactly make an object move across a drawing area without destroying the other objects came up. This was a key item since being able to do something like this is standard in just about every drawing package on the market.

Ideally, it would have been easier to have found a routine in Xlib or the Intrinsics that will take a definition of an object and allow you to drag it across a drawing area while the routine took care of not destroying the drawing. Unfortunately (in this case), Xlib and the Intrinsics are the very low-level routines of X. They only provide access to the drawing area and do not provide a routine like the one needed. Considering this, we decided to use Xlib to write routines that would do this.

When moving an object across the drawing area, one desires to erase the object from it’s old location and redraw it in a new location. Repeating this idea until the object is in it’s final place give the illusion of moving the object. A way was needed so that the object could be drawn and erased without destroying the background image in the process. The final solution to this problem turned out to be fairly simple.

This was done by drawing the objects in an exclusive-OR mode until the final location was reached. Drawing objects in this manner does alter the background when initially drawing the object, but it is restored when the object is erased.

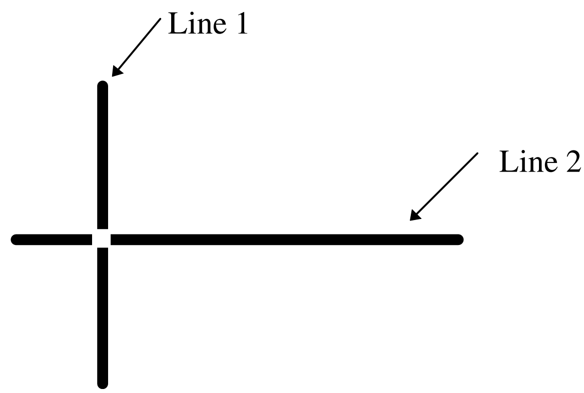

Figure 5

For example, in Figure 5 with Line 2 already on the drawing area, drawing Line 1 in an exclusive-OR mode would cause their intersection to turn white. Though this may appear to be destroying the background, when you go to erase Line 1 (once again using exclusive-OR) to redraw it in another location the hole in Line 2 will be replaced. This is the basic concept that makes dragging the objects across the drawing area possible.

Moving objects is achieved by repeating the process of drawing and erasing with the exclusive-OR mode until the object reaches it's final location and then it is drawn using an OR mode that drawn the complete object in place. This leaves the object correctly drawn. The same similar idea is used when erasing/moving text.

Summary and Conclusions

The design of CACES began with a small program that let the user draw lines on a drawing area and has since grown into a much larger editing system. This growth has required a large amount of work and learning on the part of the programmer. The system will undoubtedly be changed in the future to allow it to work in the tutoring system that is under development.

From the beginning, it was understood that CACES was to prove that such a tool could be built. Future tools of this nature will naturally do things in different ways as new ideas are found. Despite finding new ways of building tools, the major elements developed as part of CACES should be found in all of them. These will be considered the core elements of any future circuit editing system.

Future Work

Since it is understandable that all software evolves from it's beginning into a better product, CACES too will undoubtedly change in the future. Several ideas have been discussed as to some of the directions that CACES should move towards in order to make it a better system.

- Graphical editor to build object definitions
- Ability to print drawings
- Integrate CACES into a tutoring system

Each of these would only enhance the CACES system. Of the items listed, the last will undoubtedly be implemented. The first will likely be implemented at some time in the future, while the middle item would be nice, but is not considered a priority.

Appendix A

File Formats

Since many of the parameters about the CACES system are stored in files, this appendix will explain the formats of each file so that any future modification can be made easier.

Object Definitions

The files that define the objects are stored in files with a .def extension. The definition file includes not only the appearance of the object, but also the locations of the connection points that the object will have. The files used in this initial release of CACES were developed by hand using pen and created using a text editor. As stated in the Future Work section, an editor of some type (preferably graphical) would make defining these objects much easier.

Once the object definition files have been created, the program will need to know which ones to load. There is another file called draw.tools.defs which stores the file names of the objects to load into CACES. All the user will need to do to alter the objects in CACES after creating the new definition files is alter draw.tools.defs to have them loaded.

In defining an object, the program uses a coordinate system that needs to be explained. A coordinate axis is defined for each object with (0,0) being the center of the object. Using the X-Windows standard, positive X values extend to the right and positive Y values extend down.

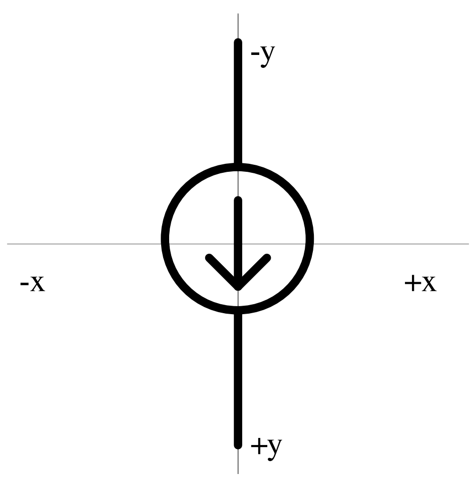

Figure 6 - Current Source

As an example, the following definition file is for the current source shown in figure 6:

```
-50 20 20 20
0 35
0 -35
6
2 -15 -15 15 15 0 0
1 0 -35 0 -15 0 0
1 0 15 0 35 0 0
1 0 -10 0 10 0 0
1 0 -10 -5 -5 0 0
1 0 -10 5 -5 0 0
```

tool03.def

Line Number

1	X-value of label	Y-value of label	X-value of value	Y-value of value
2	X-value of first connector	Y-value of first connector		
3	X-value of first connector	Y-value of first connector		
4	Number of sub elements in object definition			
5	Sub element definitions (see below)			

The first line defines the locations of the value and the label for the object. In this case, the label is located at a point (-50, 20) in relation to the center of the object which is assumed to be (0,0). The value it located at (20,20).

The next line is used to define the location of the first of two connectors associated with this object. This is because each real object represented in the circuit editor has two ends to connect wire to. As you can see, the first connector is at (0,35) and the second is at (0,-35). These points are rotated as the object is rotated so that they will line up with the drawn ends to the lines that represent the connection points.

The next number represents the number of sub elements that make up the object. This is followed by the types and locations of each of the sub elements.

The format for the lines that define the sub elements are as follows:

Type	Start X-value	Start Y-value	End X-value	End Y-value	Arc Start	Arc End
LINE=1 CIRCLE=2 BOX=3 ARC=4	integer	integer	integer	integer	Degree value to start arc	Number of degrees to extend arc

Using this format, the user can create a wide variety of objects for use with CACES. The only restriction to what can be drawn is that it must be made up of the sub elements provided.

Drawing File

When ever you press the "Save" button, the drawing that has been created will be save to whatever filename you select in the file selection box. This file is what the intelligent tutoring system will need to understand in order to model this circuit. The following file is the output file generated from the drawing in Figure 1.

Output file, representing drawing

```
3
140 310 4 0 0 0
*
24V
*
*
475 310 1 0 1 0
C1
25 uF
*
*
300 275 2 90 0 1
R1
50 ohm
*
V1
3
335 275 475 275
140 275 265 275
140 345 475 345
```

The first line of the file gives the number of objects in the drawing. The specifics about each object are then given (location, type, rotation, etc.). Next the number of lines are given as well as the points that define the lines. The general form of this file is as follows:

Line #

1	Number of objects in drawing (3 in this case)					
2	X location of object	Y location of object	Object type	rotation	show current	show polarity
3	Label String of object					
4	Value String of object					
5	Current String of object					
6	Polarity String of object					
7-16	Repeat of lines 2-6 to define other 2 objects.					
12	Number of wires in drawing (3, in this case)					
13	start x	start y	end x	end y		
14	"	"	"	"		
15	"	"	"	"		

The tutoring system should be able to use this file to parse it into another representation. This is because the objects are define as well as how they are interconnected, which is very important to any program which might try to model a circuit.

References

1. Lidtke, Doris K. ad Moursund, David, “Computers in School: Past, Present, and How We Can Change the Future,” ***Communications of the ACM***, May 1993, Vol. 36, No. 5

2. Yoshikawa, A., Shintani, M., and Ohba, Y., “Intelligent Tutoring Systems for Electric Circuit Exercising,” ***IEEE Transactions on Education***, August 1992, Vol. 35, No. 3

3. de Coulon, Frederic, Forte, Eddy, and Rivera, Jose Maria, “KIRCHHOFF An Educational Software for Learning the Basic Principles and Methodology in Electrical Circuits Modeling,” ***IEEE Transactions on Education***, February 1993, Vol 36, No. 1

4. Chowdhury, Badrul H. and Clark, Dennis E., “COPERITE - Computer Aided Tool for Power Engineering Research, Instruction, Training and Education,” ***IEEE Transactions on Power Systems***, November 1992, Vol. 4, No. 4

5. Cvetkovic, S. R., Seebold, R. J. A., Bateson, K. N., and Okretic, V. K. “CAL Programs Developed in Advanced Programming Environments for Teaching Electrical Engineering,” ***IEEE Transactions on Education***, May 1994, Vol. 37, No. 2

6. Polson, Martha C. and Richardson, J. Jeffrey *et al*, ***Foundations of Intelligent Tutoring Systems***, Lawrence Erlbaum Associates Publishers, New Jersey 1988.