

Sketch Recognition for Course of Action Diagrams

by

Kevin Stolt

Submitted to the Department of Electrical Engineering and Computer
Science

in partial fulfillment of the requirements for the degree of

Masters of Engineering in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2007

© Massachusetts Institute of Technology 2007. All rights reserved.

Author
Department of Electrical Engineering and Computer Science
September 7, 2007

Certified by.....
Randall Davis
Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by.....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Sketch Recognition for Course of Action Diagrams

by

Kevin Stolt

Submitted to the Department of Electrical Engineering and Computer Science
on September 7, 2007, in partial fulfillment of the
requirements for the degree of
Masters of Engineering in Computer Science and Engineering

Abstract

This thesis describes a software program that recognizes hand-drawn Course of Action diagrams. User input is through sketching, or a combination of sketching and speech. Course of Action symbols are recognized incrementally, and the informal sketching input is replaced with formal images of the symbols. The system uses the LADDER shape definition language to represent the geometric properties of shapes, and is capable of recognizing 327 distinct Course of Action symbols. The Intermediate Feature Recognizer is used to recognize shapes of intermediate complexity and is capable of recognizing some shapes that cannot be described using LADDER definitions. By detecting features of intermediate complexity, the system is capable of automatic error correction of some stroke segmentation errors and dealing with filled-in and multi-segment lines. The system is also able to recognize a combination of speech and sketching input of some information that can't easily be communicated through sketching alone. The system has a shape grammar to allow the sketch recognizer to conform to rules for creating Course of Action symbols. The system is also capable of "interpreting" the sketch - understanding the higher-level details of military units and actions that were sketched in the Course of Action diagram.

Thesis Supervisor: Randall Davis

Title: Professor of Electrical Engineering and Computer Science

Acknowledgments

I would like to thank my supervisor Professor Randy Davis for his advice and support throughout this project. Without his help, this thesis would not have been possible. His assistance provided excellent feedback on research ideas and kept me motivated to improve the system.

I would also like to thank Aaron Adler for always enthusiastically helping whenever I had a question or needed assistance and other members of the Design Rationale Group for their help throughout the year.

Finally, I would like to thank my family - for their constant support and encouragement.

Contents

1	Introduction	15
2	Background	19
2.1	Course of Action Recognition Systems	19
2.2	Motivations for Intermediate Features in Sketch Recognition	20
2.3	Course of Action Symbols	22
3	System Usage	27
3.1	Sketching	27
3.1.1	Creating shapes	27
3.1.2	Editing symbols	28
3.2	Multimodal Input	29
3.2.1	Symbol Naming	30
3.2.2	Editing Shapes	30
4	System Architecture	33
4.1	Sketch Recognizer	33
4.1.1	Primitive Recognizer	34
4.1.2	Intermediate Feature Recognizer	36
4.1.3	Domain Recognizer	37
4.2	COA Domain Handler	39
4.2.1	Apply COA symbolic rules to Sketch Recognizer	39
4.2.2	Interface	45

4.3	Multimodal Recognizer	48
4.3.1	Speech Input	48
4.3.2	Combining Speech and Pen Input	49
5	Intermediate Feature Recognizer	53
5.1	Limitations of LADDER System	53
5.2	Shape Detection using the Intermediate Feature Recognizer	54
5.3	Error Correction using the Intermediate Feature Recognizer	55
5.3.1	Correcting initial stroke segmentation errors	56
5.3.2	Correcting filled-in lines and multi-segment lines	58
5.4	Recognition of Intermediate Features	58
5.4.1	Dashed lines	58
5.4.2	Compound lines	59
5.4.3	Dashed Chains	60
5.4.4	Dashed Ellipses	61
6	Modifications of the LADDER System	63
6.1	Multiple Inheritance	63
6.1.1	Symbol Editing	64
6.2	Image Replacement and Composition	66
6.2.1	Preserving Scale, Translation, and Rotation	67
7	Results	73
7.1	Recognized Shapes	73
7.2	Combinatorics	76
7.3	Future Work	76
7.3.1	Future Capabilities of Multimodal Input	76
8	Conclusion	85

List of Figures

1-1	Example of a basic COA sketch	16
2-1	A complicated glyph bar found in the COA diagram program nuSketch Battlespace [5]	20
2-2	An example of a COA symbol	23
2-3	The color of a COA frame is used to indicate the affiliation of the unit	23
2-4	Specific geometric frames are used to indicate the dimension of the unit	24
2-5	Examples of COA action symbols	25
3-1	Scribbling over one shape will delete it	28
3-2	Scribbling over multiple shapes will delete each of the shapes	29
3-3	The hand cursor indicates the shape being moved.	29
4-1	System Components	34
4-2	Pen-Input Data Classified as Line Primitive Type	35
4-3	Pen-Input Data Classified as Ellipse Primitive Type	35
4-4	The LADDER shape definition of an arrow	38
4-5	Three sketched lines which may recognized as components of complex shapes	38
4-6	Labeled shapes that satisfy the constraints for the LADDER shape definition of an arrow	39
4-7	Relationship between a Sketched Shape and Objects stored in LAD- DER Recognizer	40
4-8	Relationship between a Sketched Shape and Recognized Shapes	40

4-9	Relationship between a Sketched Shape and Recognizer Shapes	41
4-10	The frame template hierarchy for a friendly unit allows any frame to be used	42
4-11	The frame template hierarchy for a friendly brigade does not allow or O echelon modifiers	43
4-12	COA Domain Handler determines the Aggressor and Defender units for a Penetrate Action	47
4-13	COA Domain Handler determines the following unit and followed ac- tion for a Follow and Support Action	48
4-14	Example COA Diagram before multimodal input	50
4-15	Two sketching inputs were received during speech input - both inputs were points	51
4-16	Result of using multimodal input to "copy" a unit	51
5-1	A rectangular frame that contains a gap	57
5-2	The IFR combines the filled-in line (highlighted in green) with the line highlighted in red	57
5-3	The grey line indicates the line recognized by the IFR from the input shown in Figure 5-2	57
5-4	A dashed line can be recognized from as few as two dashes	59
5-5	As more dashes are recognized as part of a dashed line, the endpoints of the dashed line update	60
5-6	The IFR recognizes dashes arranged in a circle as a Dashed Ellipse shape.	61
6-1	Multiple Inheritance of Symbol Construction	65
6-2	The geometric properties of displayed images may not match with in- put pen strokes	68
6-3	User strokes indicating a friendly unit are replaced by a fixed-size image	70
6-4	User strokes indicating a minefield are replaced with an image scaled to the size of the sketched minefield	70

6-5	User strokes indicating a friendly unit are replaced by a horizontal image, regardless of the orientation of the sketched rectangle	71
6-6	User strokes indicating a minefield are replaced with an image at the same orientation as the sketched strokes	72
7-1	Sketched Input to COA Design Interface	74
7-2	Displayed Output from COA Design Interface	75

List of Tables

2.1	Echelon modifiers and their symbols	25
4.1	The COA Domain Handler updates its representation of a unit as it is modified.	45
7.1	Task-Organized Icon Modifier Combinations	76
7.2	Summary of Recognized Shapes	77
7.3	Recognized COA Frames	77
7.4	Recognized COA Echelon Modifiers of Friendly Units	78
7.5	Recognized COA Echelon Modifiers of Enemy Units	79
7.6	Recognized COA Icon Symbol Modifiers of Friendly Units - 1	80
7.7	Recognized COA Icon Symbol Modifiers of Friendly Units - 2	81
7.8	Recognized COA Icon Task Organized Modifiers of Friendly Units	81
7.9	Recognized COA Icon Symbol Modifiers of Enemy Units	82
7.10	Recognized COA Action Symbols	83
7.11	Recognized COA Object Symbols	83

Chapter 1

Introduction

Sketching is a quick and efficient method for communicating information, especially spatial and geometric information [3]. One particularly appropriate use for sketching is in military Course of Action diagrams. A Course of Action (COA) diagram is created by military planners when they are formulating a battle plan. COA diagrams consist of a collection of symbols that represent military units and the actions they are to perform. COA diagrams are today created primarily with pencil and paper, or on acetate overlays on maps with grease pencils, post-its, and pushpins. Many computerized attempts at speeding up the process of COA generation have been rejected by military users, commonly because of the awkwardness of mice and menus for what is more naturally done by sketching [5]. An example of a basic COA sketch is shown in Figure 1-1.

In order to make the creation of COA diagrams effortless for the user, interaction must occur effortlessly and the interface should be invisible to the user. As suggested by Davis in [3], interacting with a software program should feel natural, informal, rich, and easy. If this goal is to be reached, the menu bars, drop-down menus, and glyph bars popular in many current programs should not be a part of the sketch interface. Instead, recognition should be the focus. A useful program in the COA domain would be able to recognize COA symbols incrementally as they are sketched based on the same geometric properties that humans use to interpret a sketch. An ideal system would allow users to sketch COA diagrams just as is done on pencil and

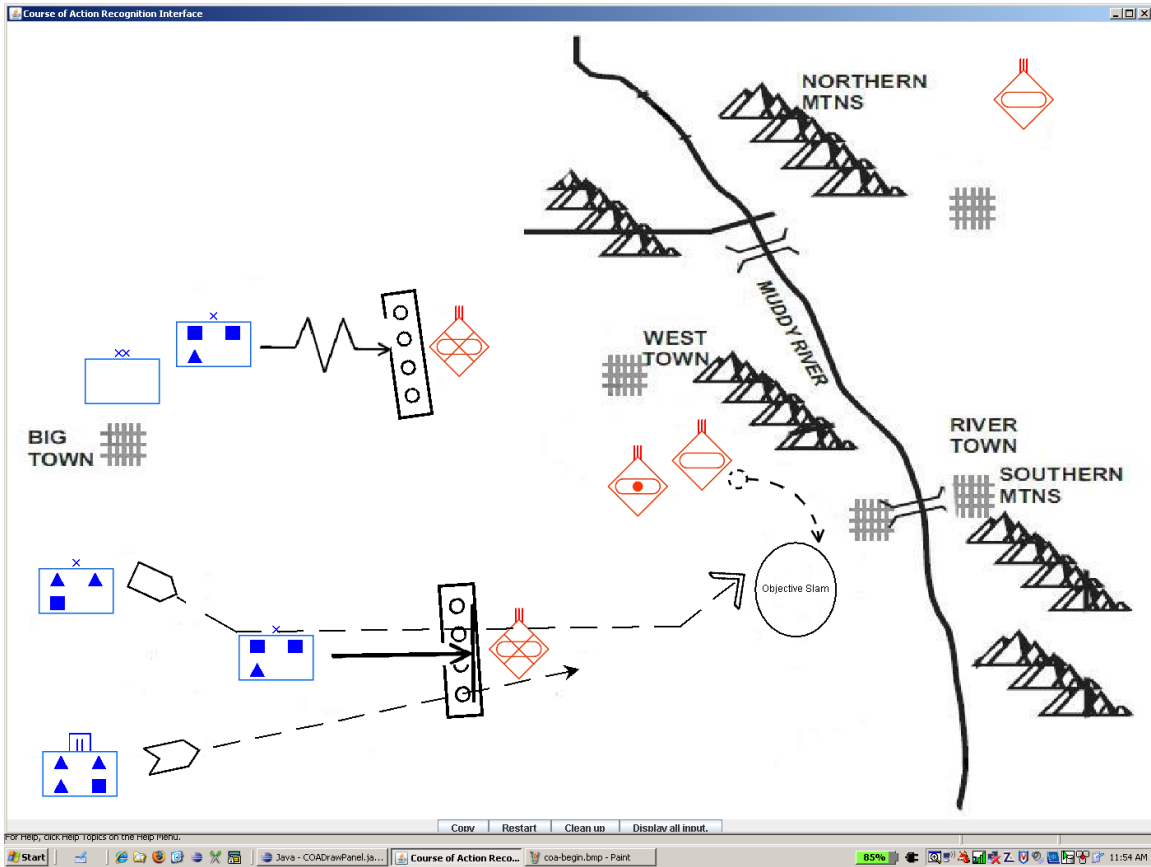


Figure 1-1: Example of a basic COA sketch

paper, but the system would be able to understand the sketch. Such a system might allow users to edit COA diagrams or simulate a battle plan from a COA diagram.

In order for a program to be able to make sense of messy, informal sketches, as shown in Figure 1-1, the system must be knowledge-based [3]. That is, the system should understand the geometry of the shapes in the domain. Hammond [6] has developed a system that interprets sketches based on geometric properties of shapes encoded with LADDER shape descriptions. LADDER is a language that can be used to describe how sketched shapes in a domain are drawn, displayed, and edited.

The LADDER shape language can be used for recognizing COA diagrams because it encodes information about the properties of shapes in the COA domain. Once LADDER shape definitions for symbols in the COA domain have been written, they can be used by a LADDER domain recognition system to recognize sketched objects.

I have created a program that allows a user to sketch a COA diagram graphically and that interprets the sketch. The program utilizes LADDER shape definitions, which are used to encode information about the geometric properties of symbols in the COA domain. Users are able to create and edit COA symbols using pen-based input. The program replaces sketched strokes with images of COA symbols, producing a neatly drawn sketch from the user's hand drawn examples. The program I developed represents an improvement over other COA software systems because it is able to interpret and understand a user's sketch as it is being drawn, similar to [2] in the mechanical engineering domain.

In addition to developing LADDER shape descriptions for the COA domain, I developed several additional components, including the Intermediate Feature Recognizer, the COA Domain Handler, and the Multimodal Recognizer. The Intermediate Feature Recognizer (IFR) recognizes shapes of intermediate complexity, following on results from studies on visual classification. The IFR is also used to recognize shapes that can't be described using LADDER shape definitions. The COA Domain Handler is used to apply COA symbolic rules to the sketch recognizer, as well as provide an interface to other systems. The multimodal recognizer combines both pen and speech input and is used to add information to the COA diagram that can't easily be communicated using sketch input alone.

Chapter 2 describes background information about other COA systems, intermediate features, and symbols of the COA domain. Chapter 3 describes how to use the software program that I developed. Chapter 4 describes the system architecture for the software program. Chapter 5 explains the Intermediate Feature Recognizer, which is used to recognize intermediate features in a sketch. Chapter 6 describes modifications made to the LADDER system, including support for multiple inheritance and image replacement. Chapter 7 discusses results and shows examples of using the system to create a COA diagram.

Chapter 2

Background

2.1 Course of Action Recognition Systems

Computerized COA systems have typically been rejected primarily because the interface seemed unnatural to users. Some systems relied on a complicated user interface, which added complexity to creating COA diagrams. For example, nuSketch Battlespace (nSB) [5] relies on a glyph bar to create units and drop-down menus for unit properties and actions. While this avoids the problem of recognition, it also creates a complex and inefficient interface. The glyph bar is shown in Figure 2-1. Instead of sketching, users create symbols by clicking on the appropriate icon on the glyph bar and adding details about the icon through the appropriate drop-down menu. In nSB, 294 distinct friendly unit symbols and 273 distinct enemy unit symbols are represented in the glyph bar. While trying to speed up the creation of COA diagrams, the user interface of nSB adds complexity and complicates the sketching task, making it difficult to interact with the program.

We believe the goal should be to design software with as few restrictions as possible on the user. As described in [1], such a system would allow the user to sketch freely, without modifying their drawing style, yet still be able to interpret the user's drawing and allow the user to interact with it.



Figure 2-1: A complicated glyph bar found in the COA diagram program nuSketch Battlespace [5]

2.2 Motivations for Intermediate Features in Sketch Recognition

When a human looks at a picture or sketch, they perform object classification quickly and easily, easily recognizing familiar objects in the picture. One of the questions from visual processing research asks which features of objects in the picture allow for the best classification. Ullman [8] states that human visual processing begins by using simple local features, and then subsequently representing the image in terms of larger and more complex features. He found that the feature size that is most optimal for object classification is that of intermediate complexity.

Ullman determined that features of intermediate complexity (IC) are more informative than features that are very simple or very complex. Features of intermediate complexity serve as building blocks of a class. Ullman states that there are two factors which explain the superiority of intermediate-size features: specificity and relative frequency. A large and complex feature can provide reliable indication of the presence of a class, but it is not generally representative of the class as a whole; new examples in the class would likely not match with the large feature. Therefore, large and complex features may be too specific and too infrequent to classify objects in the class. Small and primitive features have a much higher likelihood of occurring, but their presence is also more likely to occur outside of the class as well. Therefore, small and primitive features may be too general and occur too frequently to be an indicator for a specific class of objects. Features of intermediate complexity are more likely to generalize broadly across a class of objects, but they are also complex enough to occur rarely outside the class of objects.

While Ullman's research showed that intermediate complexity features contain the most information, it is also important to identify what information they contain. According to Ullman, the features of a common classification can be represented in terms of simpler fragments. These fragments combine together hierarchically to produce an intermediate feature based on low level features. Consider features used in identifying a human face. An intermediate feature might cover the nose and mouth region. One low-level feature that is part of the intermediate complexity feature is a vertical region indicating a nose. Another low-level feature is two horizontal regions that are stacked indicating lips. The entire intermediate complexity feature consists of the two simpler features and their arrangement - a nose feature vertically centered above a mouth feature. Intermediate features with this type of representation were shown to perform better in visual classification of images than both larger and smaller features. The important thing to note is that intermediate features are described in terms of the primitive features of which they're composed and the primitive features' arrangement within the intermediate complexity feature.

The research conducted by Ullman was performed using images. Intermediate

features are relevant to sketch recognition because a sketching environment contains several sources of noise. One main source of noise in a pen-based environment is the users. The system can identify where the pen is on the screen more finely than the user can control it. If the user attempted to draw a perfectly straight line, the system would easily be able to identify waves, fluctuations, and jitters in the line produced by the user. A somewhat smaller source of noise in a sketching environment is the precise location of the pen tip on the screen. The head of the pen is larger than the location of the pen capture by the system. The precise location that the user intended is somewhat obscured by the size of the pen head. This doesn't cause recognition problems for users, since this detail is on a very small scale, and in many cases may not even be perceivable. Another source of noise results from the informal nature of the sketching environment. Often, users do not try to draw precise shapes in a sketch. Drawing two perpendicular lines that share an endpoint is very difficult, since the lines must be have precisely a 90 angle and must be precisely coincident. Since a sketching environment is not precise, this source of noise is quite common. A system that recognizes objects in sketches needs to be able to deal with these sources of noise, to determine the users' intent for object recognition.

Intermediate features can be used to recognize shapes even when a sketch is informal. Current systems begin this process by stroke segmentation and primitive object detection. However, these methods still are dependent on individual pen strokes, so they may not capture the features of a sketch at an intermediate level. Some common examples of shapes that aren't able to be recognized using this system are dashed lines, filled-in lines, and multi-segment lines. These shapes could be recognized using an intermediate feature recognizer, which would combine structural details of primitive objects to recognize higher-level shapes.

2.3 Course of Action Symbols

The United States military has created a symbolic language to visually represent various aspects of military operations, as described in [4]. Symbols can be constructed

to represent a variety of military units, equipment, control measures, installations, and operations. Military commanders and their staffs use these symbols to create situation maps, overlays, and annotated aerial photographs for all types of military operations. These diagrams are known as Course of Action diagrams. Course of Action (COA) diagrams represent operational plans and orders, as well as known locations of friendly, hostile, and neutral units and installations. A single graphical display of symbols can describe an entire operational picture.

There is a corresponding doctrinal meaning for each graphical symbol in the language, which is standardized across all branches of the military. Symbols representing units contain detailed information about various aspects of the unit including its strength, size, branch, affiliation, dimension, and composition. Operation symbols represent tasks to be performed. Because the symbolic language is standardized, operational information can be passed quickly between military units.

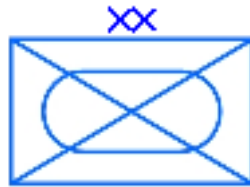


Figure 2-2: An example of a COA symbol

Each military symbol that represents a unit in the COA domain can be constructed from a frame. Additional information about the unit is represented by text modifiers, graphic modifiers, and the color of the unit frame. In the symbolic language, frames

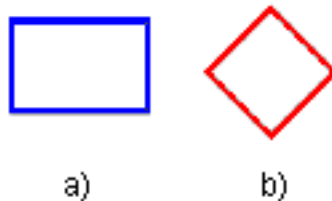


Figure 2-3: The color of a COA frame is used to indicate the affiliation of the unit

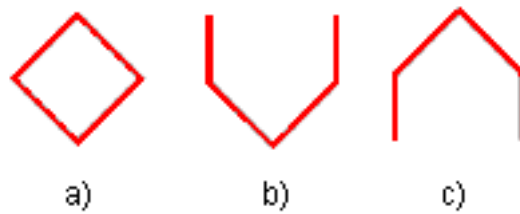


Figure 2-4: Specific geometric frames are used to indicate the dimension of the unit are the geometric border of the symbol. The frame serves as the base of a symbol to which additional modifiers can be added. In addition, the frame conveys information corresponding to the affiliation, dimension, and status of a unit.

- Affiliation - The frame of a symbol conveys information about the affiliation of the symbol. This is conveyed by the geometric structure of the frame. Each affiliation also has a corresponding color, so friendly and hostile units can be clearly distinguished in a course of action diagram. Figure 2-3a shows a friendly unit (colored blue) and Figure 2-3b shows an enemy unit (colored red).
- Dimension - The frame conveys whether the symbol is used to represent a land, sea surface, sub surface, air and space, or unknown dimension. Three dimensions of an enemy unit - land, sub-surface, and air - are shown in Figure 2-4 a, b, and c, respectively.
- Status - The frame also indicates the status of a symbol. Frames composed of solid lines indicate the present location of a unit. If the frame is composed of dashed lines, it represents the pending or suspected location of a unit.

The symbol icon can be drawn inside of the frame, to express the function and role of the unit represented. Text and graphic modifiers can be added to specific locations of the frame to represent additional details about the unit. One such modifier that can be added is the echelon, which is centered directly above the frame. Some echelon symbols and their meaning are shown in table 2.1.

In addition to symbols representing military units, which are constructed by frames and frame modifiers, there are also symbols that represent military actions.

Squad	•
Section	••
Platoon	•••
Company	
Battalion	
Regiment	
Brigade	×
Division	××
Corps	× × ×

Table 2.1: Echelon modifiers and their symbols

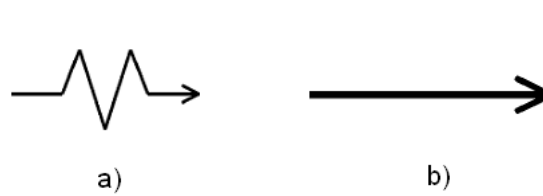


Figure 2-5: Examples of COA action symbols

Figure 2-5 shows examples of action symbols in the COA domain: Figure 2-5a is a fix action and Figure 2-5b is a penetrate action. Often, action symbols express battlefield tactics and indicate what operations a unit is to perform. These action symbols are especially useful when planning a battle because there is a corresponding doctrinal meaning as to what action a military unit should take in a battle.

Chapter 3

System Usage

This chapter describes the program I developed that can be used to develop a COA diagram. Users provide information about the operational picture through sketching or a combination of sketching and speech input. The program provides feedback to the user by transforming pen strokes into formal COA symbols.

3.1 Sketching

Sketching is the primary method of input to the COA design interface. Sketch input can be divided into two types of pen strokes - those used to create shapes and those used to edit shapes.

3.1.1 Creating shapes

The shapes displayed in Chapter 7 can be recognized by the software program. Shapes are recognized based on their geometric properties, so the user can draw strokes in any order and at any scale. Once a shape has been recognized, it is replaced by the image representing the COA shape. This gives feedback to the user about whether their most recent sketched input was interpreted correctly. If the sketch was not interpreted correctly, this gives the user the option to fix the symbol or delete it.

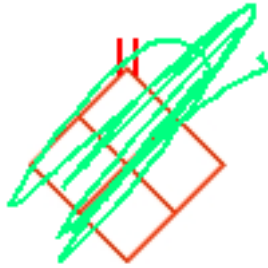


Figure 3-1: Scribbling over one shape will delete it

3.1.2 Editing symbols

Users can also edit the diagram: symbols can be deleted, moved, and copied using editing pen strokes.

Deletion

If the user determines that an object is not recognized correctly, they can erase all of the object, then redraw it. This allows the user to correct errors made by the recognition system: shapes can be deleted by scribbling over them. After a scribble pen stroke is completed, any shapes that have been scribbled over are removed from the drawing panel, and the scribble pen-stroke disappears from the drawing panel.

Figure 3-1 shows an example of a scribble stroke, shown in green over the red unit indicating an enemy infantry battallion. A single scribble can delete multiple shapes, as shown in Figure 3-2. In this example, a scribble (shown in green) is drawn over a minefield (shown in black) and a friendly mechanized artillery brigade (shown in blue). The scribble deletes both the minefield and friendly unit.

Moving

To move shapes, the user touches the pen to the screen over the shape for about half a second, after which the cursor changes to a hand (Figure 3-3). Once the cursor has changed, the pen tip may be moved around the drawing panel and any shapes located

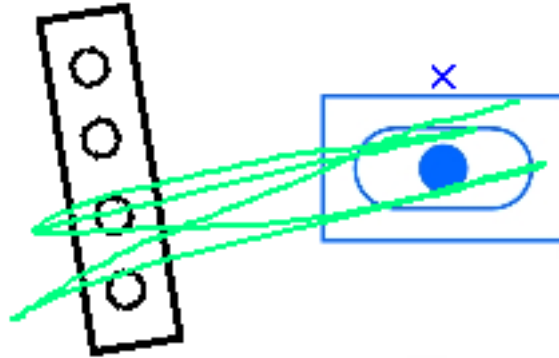


Figure 3-2: Scribbling over multiple shapes will delete each of the shapes

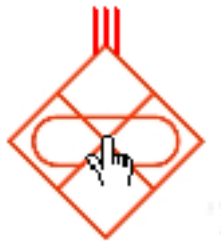


Figure 3-3: The hand cursor indicates the shape being moved.

directly beneath the cursor follow. In Figure 3-3, moving the pen around will cause the enemy unit to follow.

3.2 Multimodal Input

The user may also interact using a combination of sketching and speech. Speech interaction allows the user to communicate information to the system that may be difficult or impossible to communicate through pen-based interaction. To use speech input, the user touches the pen-tip to the “Talk” button to indicate the beginning and end of speech input.

3.2.1 Symbol Naming

Certain symbols in the COA domain can be named through the COA Design Interface using multimodal interaction. Currently, only “Objective” areas can be named. In order to assign a name to an area, the user follows the commands listed below.

1. Press “Talk” button.
2. Say “This is <symbol-name>” while clicking once on screen (to indicate the object to be named), where <symbol-name> is the name of the symbol.
3. Press “Talk” button.

3.2.2 Editing Shapes

The editing capabilities of multimodal interaction are also possible through pen-based interaction, but both options are available and it is up to the user to decide which form of interaction feels more natural.

Move/Copy (Option 1):

1. Press “Talk” button.
2. Say “Copy [Move] this unit [object] here” while clicking twice on screen, once to indicate location of object to be moved [copied] and a second time to indicate the new location of the object.
3. Press “Talk” button.

Move/Copy (Option 2):

1. Select item on screen with pen-input.
2. Press “Talk” button.
3. Say “Copy [Move] selected unit [object] here” while clicking once on screen (to indicate new location of object).
4. Press “Talk” button.

Delete (Option 1):

1. Press “Talk” button.
2. Say “Delete this unit [object]” while clicking once on screen (to indicate location of object to be deleted).
3. Press “Talk” button.

Delete (Option 2):

1. Select item on screen with pen-input.
2. Press “Talk” button.
3. Say “Delete selected unit [object]”.
4. Press “Talk” button.

Chapter 4

System Architecture

There are three main components of the COA interface: the sketch recognizer, the COA Domain Handler, and the COA Multimodal Recognizer. The sketch recognizer is capable of recognizing sketched shapes in the COA domain. The COA symbol domain is compositional - multiple modifiers can be added to symbols. The COA Domain Handler ensures that only valid combinations of modifiers are recognized by the sketch recognizer. The COA Domain Handler also provides an interface to other systems. The COA Multimodal Recognizer combines pen and speech input, allowing the system to receive details through voice input that might not be easily sketched.

4.1 Sketch Recognizer

The sketch recognizer consists of three components, the Primitive Recognizer, the Intermediate Feature Recognizer, and the Domain (LADDER) Recognizer. As pen-input data is captured by the computer, it is passed to the primitive recognizer, which analyzes and classifies individual pen-strokes as lines, ellipses, points, polylines, and scribbles. Once an initial classification of these strokes has been completed, primitive objects representing lines, ellipses, etc. are passed to the Intermediate Feature Recognizer and Domain Recognizer. The Intermediate Feature Recognizer is used to recognize shapes of intermediate complexity that are drawn with one or more strokes. Recognized shapes from both the Primitive Recognizer and the Intermediate Feature

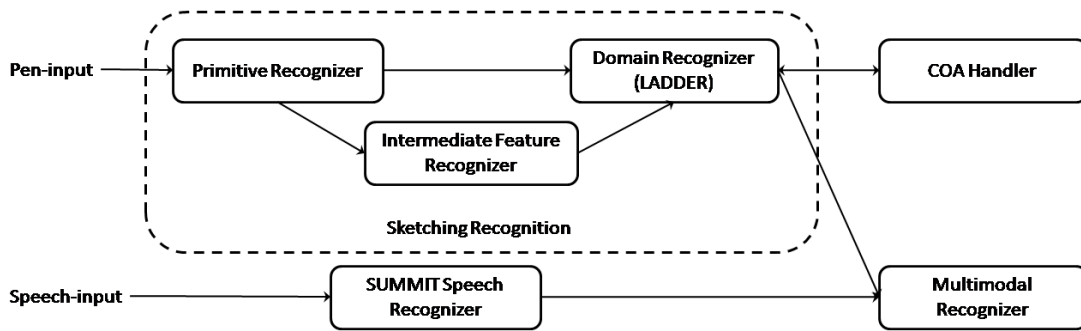


Figure 4-1: System Components

Recognizer are combined in the Domain Recognizer in order to recognize complex shapes in the domain. Figure 4-1 shows the architecture for system components.

4.1.1 Primitive Recognizer

Pen data collected from the hardware contains information about the vertical and horizontal position of the pen-tip on the screen, as well as the pressure of the pen-tip on the screen, and the time the input was detected. Each time the pen touches the screen and is removed is referred to as a pen stroke. Each pen stroke consists of one or more points.

The primitive recognizers determine whether the pen stroke can be classified as an ellipse, line, point, polyline, or scribble. There is an independent classifier for each type of primitive. If the classifier determines that the stroke can be classified as one of these primitives, a primitive object of that type is created and passed to the domain shape recognizer. Figure 4-2 shows a sequence of points collected by the pen-hardware have been classified as a line and are displayed by the system as a line. Figure 4-3 shows a pen stroke and its resulting classification and display by the system as an ellipse primitive shape. Once a primitive object is recognized, only important reference points are kept. For example, only the endpoints of a line are kept, while the input points used to classify the line are discarded.

It is possible for a single stroke to be classified as multiple primitives, provided the

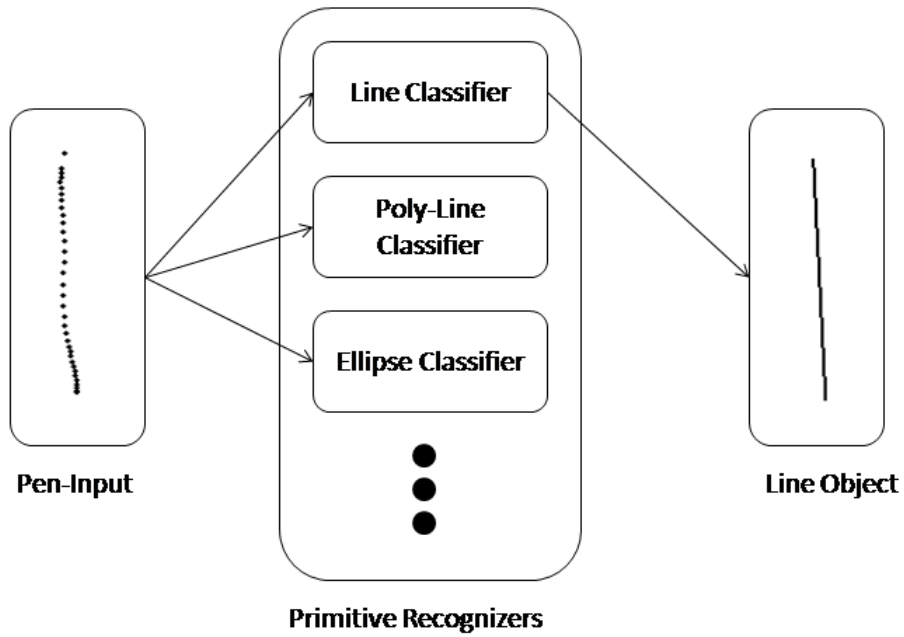


Figure 4-2: Pen-Input Data Classified as Line Primitive Type

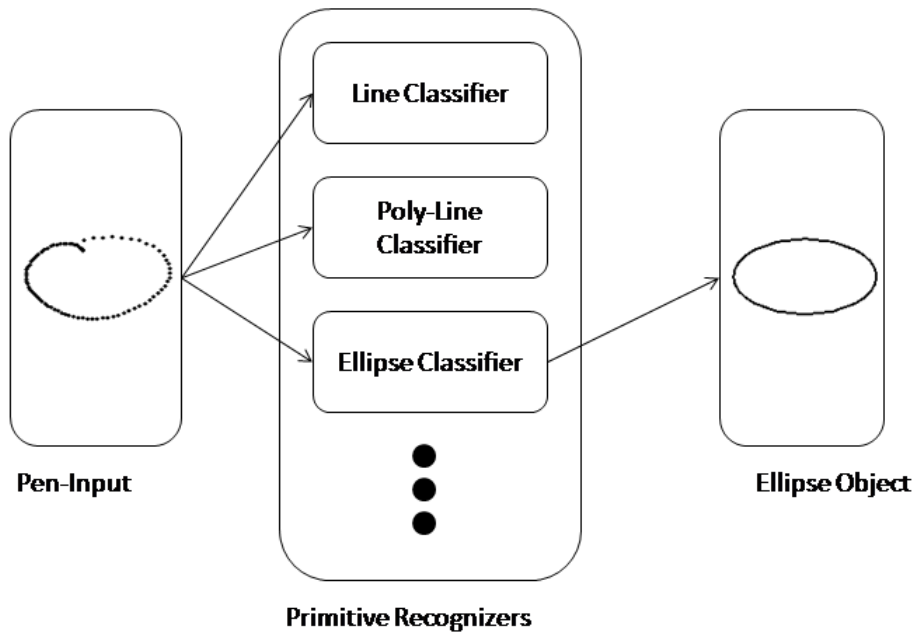


Figure 4-3: Pen-Input Data Classified as Ellipse Primitive Type

pen stroke meets the geometric requirements for each primitive shape. In that case multiple primitive objects for a single pen stroke are created and sent to the domain recognizer.

Each primitive object created when a pen stroke is classified contains details about the features of the sketched shape. In the case of an ellipse, the ellipse object contains details about the height, width, and center of the ellipse. A line object would include coordinates for each endpoint of the line, as well as its length and slope. The features associated with each primitive type are used in recognition of domain shapes, and this data is also sent to the domain recognizer when a primitive object is created.

Each recognized shape in the system contains information about which subcomponents the shape is composed of. For primitive objects, the subcomponent is the stroke. As noted earlier, each stroke is analyzed by all primitive recognizers and multiple primitive classifications of a stroke are allowed. Therefore, a single stroke may be classified as multiple primitive types, resulting in several primitive objects being created and sent to the Domain Shape recognizer. However, the domain shape recognizer allows only one of these pen stroke interpretations to be used as a component of more complex shapes. Each pen stroke has a unique ID value, and each value may be used to form at most one other domain shape.

The ellipse, line, point, polyline, and scribble primitive classifiers are used to interpret a single pen stroke. The polyline recognizer is used to segment a stroke into line components. Segmentation is determined by using speed and curvature data obtained by analyzing the pen stroke, as in [7]. The resulting line segments of the polyline are created as line primitives and sent to the domain shape recognizer.

4.1.2 Intermediate Feature Recognizer

The intermediate feature recognizer analyzes the primitive objects created by the Primitive Recognizer. If there is a collection of these primitive objects that form a shape of intermediate complexity, this information is passed to the domain shape recognizer. Therefore, there can be multiple interpretations of a single pen-stroke: (1) the low-level classification provided by the primitive recognizers, and (2) the interme-

mediate complexity feature(s) the stroke is part of. While the primitive shape recognizer recognizes primitive objects with a single stroke (low-complexity objects), the intermediate feature recognizer can detect shapes of intermediate complexity based on their geometric properties. The result of the intermediate feature recognizer is an alternate interpretation of a collection of single-stroke primitive objects. This collection is input to the domain shape recognizer.

Recognizing shapes of intermediate complexity can have several benefits to the system. The intermediate complexity shape recognized by the intermediate shape recognizer may allow automatic correction of some stroke segmentation errors, and the ability to recognize filled-in and multi-segment lines. While the domain shape recognizer may receive possible interpretations of a single stroke from both the primitive shape recognizer and the intermediate feature recognizer, only a single final interpretation is chosen based on which other shapes have been drawn. The Intermediate Feature Recognizer is discussed in more detail in Chapter 5.

4.1.3 Domain Recognizer

Using the LADDER shape definition builder, a LADDER shape definition was created for each symbol in the COA domain. The LADDER shape definition describes how a given shape should be recognized, displayed, and edited in the Domain Recognizer. Each LADDER shape definition contains a list of components of the shape being defined and a list of constraints that must be satisfied among the components. The task of the domain recognizer is to “recognize” shapes that have been sketched.

The collection of recognized shapes is stored in the Visible Shape Collection (VSC). Whenever a shape is added to the VSC, the domain recognizer checks to see if it can combine with other shapes in the VSC to produce a more complex shape. A shape is “recognized” by the domain recognizer if: (1) all of the components in its LADDER shape description are in the VSC, and (2) all of the constraints listed in the LADDER shape description for the shape are satisfied by the components. Once a shape is recognized by the domain recognizer, all of its components are removed from the VSC and replaced by the recognized shape.

```

(define shape Arrow
  (components
    (Line shaft)
    (Line headOne)
    (Line headTwo))
  (constraints
    (coincident shaft.p1 headOne.p1)
    (coincident shaft.p1 headTwo.p1)
    (coincident headOne.p1 headTwo.p1)
    (equal-length headOne headTwo)
    (acute-meet headOne shaft)
    (acute-meet shaft headTwo))
  ...
)

```

Figure 4-4: The LADDER shape definition of an arrow



Figure 4-5: Three sketched lines which may be recognized as components of complex shapes

An example of a LADDER shape definition, the definition of an arrow shape, is shown in Figure 4-4. There are three required components of the shape and six constraints that must be met for an arrow to be recognized. Figure 4-5 shows three lines that have been added to the VSC. Each of the three shapes in the VSC is a line, which satisfy the components necessary to form an Arrow. The Domain Recognizer attempts to pair LADDER components with shapes in the VSC. In this case, there is one pairing that will allow the constraints of the LADDER arrow definition to be met, shown in Figure 4-6. The new shape, an arrow is added to the VSC, and its components (the three lines) are removed from the VSC.

The LADDER shape definition and recognition system is hierarchical. The most primitive shapes are those shapes sent to the domain recognizer from the primitive recognizer. Primitive shapes and other domain shapes may combine to form more complex domain shapes, as described in LADDER shape definitions. Recognition is incremental - the entire COA diagram does not need to be sketched - only those

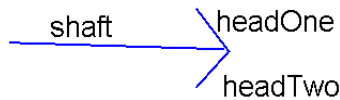


Figure 4-6: Labeled shapes that satisfy the constraints for the LADDER shape definition of an arrow

components of the shape being recognized. This is particularly appropriate for the Course of Action domain because many of the symbols can be described hierarchically.

4.2 COA Domain Handler

The COA Domain Handler has two main functions: (1) assist the sketch recognizer in following rules for constructing symbols in the Course of Action domain, and (2) “interpret” the sketch and provide an interface from the sketch recognizer to other systems.

4.2.1 Apply COA symbolic rules to Sketch Recognizer

The first function of the COA Domain Handler is to restrict recognition of some shapes by the sketch recognizer to follow the rules of drawing symbols in the COA domain. This is also closely related to the issue of multiple inheritance of COA shapes (see Section 6.1). As is discussed in Section 6.1 on multiple inheritance, each modifier is recognized in the context of the frame of a unit. A COA symbol may have many modifiers, but only is allowed one of each type of modifier. The following example demonstrates how the sketch recognizer can correctly recognize multiple modifiers for a unit, but when combined make the COA symbol invalid.

Once the frame has been sketched and recognized, the representation for the unit frame remains in the visible shape collection and can be combined with other shapes. Because the frame is in the VSC, modifiers can be added to the frame. As shown in Figure 4-7, a company modifier and an infantry modifier have been added to the frame for the unit. For the single COA symbol, the VSC contains

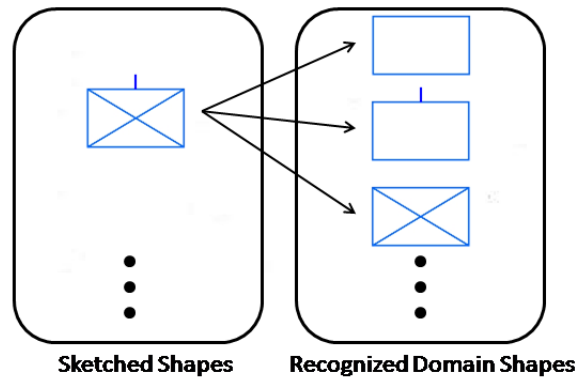


Figure 4-7: Relationship between a Sketched Shape and Objects stored in LADDER Recognizer

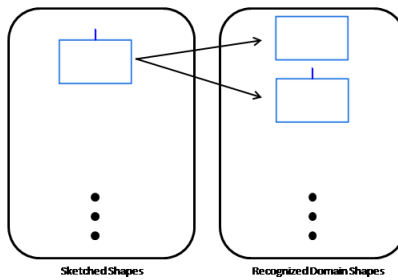


Figure 4-8: Relationship between a Sketched Shape and Recognized Shapes

three recognized LADDER shapes: the unit frame, the echelon modifier, and the icon modifier. Because these LADDER shape representations remain in the VSC, they can each be modified further. The echelon modifier (a company) can combine with another company symbol to create a batallion echelon modifier. The icon modifier (infantry) can be modified by adding an ellipse in the center of the frame indicating that the unit is armor. The armor and infantry symbol together are recognized as a mechanized infantry unit. In addition, the unit frame can be modified by adding additional modifiers (although the frame will remain in the VSC).

The following example shows a problem that might arise with allowing the frame to remain in the VSC. In Figure 4-8, the symbol for a friendly company has been drawn. The VSC contains two shapes after recognition of the shape in Figure 4-8: an echelon modifier (a company), which can be used to recognize a batallion, and

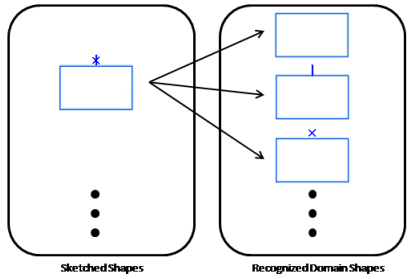


Figure 4-9: Relationship between a Sketched Shape and Recognizer Shapes

the frame itself. The frame remains in the VSC to allow recognition of additional modifiers to the frame. Figure 4-9 demonstrates a situation that may occur if the user then mistakenly decides to draw a brigade echelon modifier on the same frame. Because the frame remains in the recognizer, the sketched brigade modifier (“X”) can be combined with the frame to produce a friendly brigade because the necessary components and geometric constraints for the friendly brigade symbol are met. Both echelon modifiers - the company and the brigade - are valid in the context of the frame of a unit. However, according to the symbolic language for COA symbols, each frame may have at most one echelon modifier, hence the symbol shown in Figure 4-9 is not valid.

We use frame templates to solve this problem. Frame templates serve as a grammar to enforce the rules of constructing COA shapes. Instead of recognizing the modifier in the context of a frame, modifiers are recognized in the context of frame templates. Unit modifiers are recognized in the sketch only if the frame template exists in the VSC that corresponds to the unit modifier. The COA Domain Handler is responsible for adding and removing the frame templates from the sketch recognizer to enforce COA symbol composition rules. The use of frame templates allows the restriction of recognition of shapes to follow the COA symbolic language. All shape definitions for unit modifiers reference a specific template for a frame rather than the unit frame itself.

The frame template hierarchy for a friendly unit is shown in Figure 4-10. There are two types of templates - echelon and icon. There are three echelon templates (|, X, and

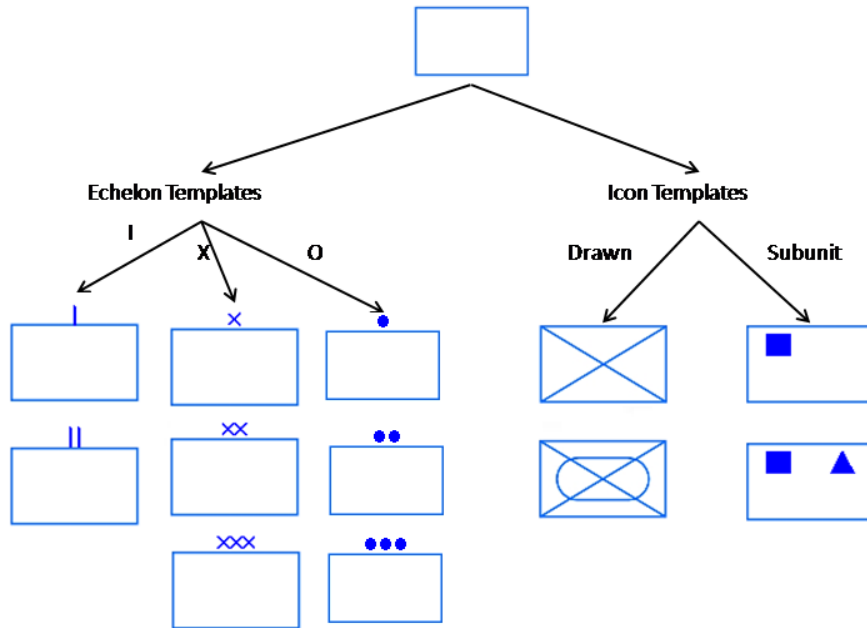


Figure 4-10: The frame template hierarchy for a friendly unit allows any frame to be used

O) and two icon templates (drawn and subunit). Examples of modifiers that use each of these templates is shown in Figure 4-10. When a modifier is recognized, the frame template hierarchy may change. Figure 4-11 shows the frame template hierarchy for a friendly brigade. Figure 4-11 shows that the | and O echelon templates have been removed, while leaving the icon templates unaffected. Additional X modifiers and either icon modifier can be recognized for the symbol, but | and O echelon modifiers are not recognized. When a template of one type has been recognized, the other templates for that type are removed from the recognizer. However, templates for different frame modifiers are unaffected. This approach allows frame modifiers to be sketched in any order, while restricting recognition by the sketch recognizer to follow the course of action symbolic language.

By adding and removing frame templates from the VSC, the COA Domain Handler influences which modifiers may be recognized, which enforces COA symbol composition rules. Frame templates are added to the VSC whenever a frame is recognized by

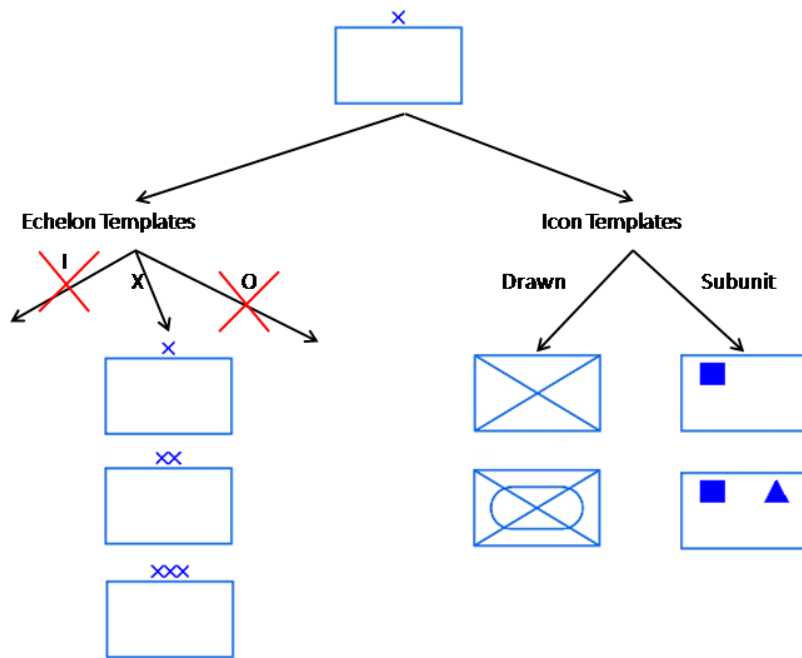


Figure 4-11: The frame template hierarchy for a friendly brigade does not allow | or O echelon modifiers

the sketch recognizer (as specified by the template grammar), which allows recognition of frame modifiers. Frame templates are deleted from the VSC when frame modifiers are recognized (as specified by the template grammar). The template grammar is used to specify which templates are added and deleted by the VSC.

As shown in Table 2.1, echelon modifiers consist of either “|”, “X”, or “O” symbols, but these symbols are never mixed. There are three possible echelon templates - the “|” template, the “X” template, and the “O” template. These three templates represent the three base shapes for echelon modifiers. Adding the first “|”, “X”, or “O” of an echelon modifier to a unit frame will cause the COA Domain Handler to remove the two unused echelon modifiers. The “|” template can be used to form company, battalion, and regiment symbols, the “X” template can be used to form brigade, division, and corps symbols, and the “O” template can be used to form squad, section, and platoon symbols. Once the first echelon modifier is added to a symbol (|, X, or O), recognition of additional echelon modifiers to the same frame are restricted to those of the same type (|, X, or O), as shown in Figure 4-10.

There are two icon templates available for friendly units (drawn symbol and task-organized subunit) and one for enemy units (drawn symbol). The icon modifier of a frame represents information about the function of the unit and can be drawn for both friendly and enemy units. For friendly units, the function of a unit can alternately be displayed using a task-organized icon. The icons in a task-organized icon give additional information about the composition of the unit. Symbols for a task-organized icon modifier and a icon symbol can't be combined in the same frame according to rules for creating COA symbols. Therefore, once an icon modifier has been recognized that is either a task-organized icon modifier or a icon symbol, the other icon template is removed from the recognizer. This allows recognition to be restricted to one of the two types of icon modifiers after the first icon modifier for a frame has been recognized.

The system currently handles two types of modifiers (echelon and icon), although it can easily be extended to handle additional types of modifiers.






COA Symbol:					
Affiliation:	Friendly	Friendly	Friendly	Friendly	Friendly
Echelon:	Unknown	Company	Company	Battalion	Battalion
Function(s):	Unknown	Unknown	Infantry	Infantry	Mechanized Infantry

Table 4.1: The COA Domain Handler updates its representation of a unit as it is modified.

4.2.2 Interface

The COA Domain Handler is an interface between the sketch recognizer and other systems. It “interprets” the sketch: combining modifiers that describe each unit and determining relationships between sketched symbols that accurately describe the operational picture. The data transmitted from the the sketch recognizer to the COA Domain Handler consists of unit frames, individual modifiers to these frames and individual action symbols in the COA domain. In order to avoid issues dealing with the large number of possible shape combinations in the COA domain, the shape recognizer does not combine different types of modifiers together nor does it associate actions with the units involved. These tasks of relating sketched symbols are performed by the COA Domain Handler.

When COA symbols representing unit modifiers have been recognized by the sketch recognizer, the COA Domain Handler updates its internal representation of that unit. The internal representation for COA symbols is an XML format developed by Draper Laboratory. The information encoded for each unit includes the unit’s strength, possible subunits, functions, and location. Action details include which units are involved in the action and the status of the action. While the locations of units are single points, the locations of minefields and objectives are areas, represented by a series of points representing the boundary of the region.

Units

Each COA symbol for a unit is represented by multiple objects in the VSC - the frame and any modifiers of that frame. The COA Domain Handler has one representation for each unit, regardless of the number of modifiers, and updates this representation as new modifiers are recognized and added to the VSC. As shown in Table 4.1, the COA Domain Handler updates information about a unit as modifiers are added to a COA unit symbol.

Information about a single unit comes from all the sketched objects that share the same frame. However, determining which units are involved in which actions must be done by other means. When COA symbols representing actions have been recognized, the COA Handler analyzes the geometry of the sketch to determine which units are involved in the action.

Aggressor/Defender Actions

Some actions consist of an aggressor and a defender, examples of which are the “Penetrate” and “Fix” actions. For these actions, the sketched shape often consists of an arrow-like shape. The shape of the arrow shaft may differ (to differentiate between actions), but the common properties are that the aggressor and defender can be determined based on the position of the tail of the shaft, the position of the head of the shaft, and the direction the arrow is pointing. A common LADDER shape definition of “tail” and “head” are used in all shape descriptions for actions of the type aggressor/defender. This allows the COA Domain Handler to locate the endpoints of the arrow and determine the direction the arrow is pointing. The COA Domain Handler has access to all recognized units and their location in the frame. Using the location of the referenced points for “head” and “tail” in the sketched shape, the COA Domain Handler determines the likely aggressor and defender based on the angle of the arrow shaft, the location of these points, and the location of all sketched units.

Figure 4-12 shows a COA diagram with four units (labeled 1-4) and a penetrate action, with the head and tail of the shaft labeled. The points at the head and tail

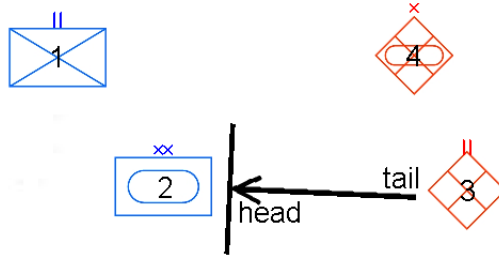


Figure 4-12: COA Domain Handler determines the Aggressor and Defender units for a Penetrate Action

are used to determine which direction the arrow is pointing. The unit nearest the head location and most directly in the path of a ray from the tail extending through the head is called the defender. This is unit “2” in Figure 4-12. The aggressor is the unit nearest the tail location and most directly in the path of a ray from the head extending through the tail (in Figure 4-12, unit “3”). The internal model of the aggressor/defender action is updated to refer to the detected aggressor and defender units.

Follower/Followed Actions

The COA Domain also contains actions which indicate that a unit should follow another tactical operation, such as the “Follow and Assume” and “Follow and Support” actions. These actions contain a base symbol (indicating the type of action and beginning location of the action), as well as a path which indicates related actions. The LADDER shape description for these shapes similarly indicates a base shape and a dashed or solid line called a “path”. The line indicating the path of the action may contain multiple segments or waypoints. For “Follow and...” actions, the system selects as the action to be followed the action that is the shortest distance from the “path”. The unit doing the following is determined by the closest unit to the base of the “Follow and ...” Action. Once the COA Handler has been determined the related follower unit and followed action, the internal model is updated to refer to the follower unit and followed action.

Figure 4-13 shows a COA diagram with six units (labeled 1-6) and three actions

at the MIT Computer Science and Artificial Intelligence Laboratory. In the SUMMIT system, segments of the audio signal are matched against a library of small units of sound called phonemes. The sequence of matches is used to produce a model of which word parts have been spoken. Humans often rely on the context of audio fragments and their own understanding of language in order to clarify speech and the SUMMIT system does as well. As part of this project, a language model for the COA Domain was developed. The COA language model and model of spoken phonemes are used to generate an n-best list of candidate sentences, each of which is accompanied by a score. This n-best list is the output of the SUMMIT system for a given sequence of acoustical signals.

4.3.2 Combining Speech and Pen Input

When there is input from multiple modalities, the COA Multimodal Recognizer combines the output from the sketch recognizer and the SUMMIT Speech system to determine the user's intent. The output from the speech recognizer, an n-best list of candidate sentences, and the output from the sketch recognizer, one or more sketched shapes, are compared by the COA Multimodal Recognizer to determine which actions to perform.

Different parts of recognized speech can be associated with a particular "tag". Each candidate sentence in the n-best list consists of one or more tags. The speech input is mostly command based, so command tags are of the form "command-type", where type refers to the type of command the user wishes the system to perform. Each type of command tag is associated with an expected input from the sketch recognizer. The expected input is a type of shape (or shapes) depending on how many inputs are expected for a given command. For example, the expected input for the move operation is a point (to select the new location of a symbol). If the expected type of input from the sketch recognizer matches with the given command of the n-best list, then the command is processed based on the input stroke. If the expected type of input from the sketch recognizer does not match the speech command, then the system doesn't do anything in response to the speech input.



Figure 4-14: Example COA Diagram before multimodal input

The following example demonstrates the process of combining speech and pen input. Figure 4-14 shows the sketched shapes at the beginning of the multimodal input command. After the user presses the “Talk” button to indicate speech input, the multimodal input begins. The user states “Copy this shape here.” At the same time, two pen strokes are received by the sketch recognizer (which are both classified as points), shown in Figure 4-15. The user presses the “Talk” button again to indicate the completion of multimodal interaction. After processing the speech input, the SUMMIT system sends the multimodal recognizer the tag “command-copy”. This is used to indicate that the user wishes to copy a shape. The expected input for a command-copy tag from the sketch recognizer is two points (to indicate the existing shape to copy and the new location) or one point (if a shape is already selected, the single point indicates the location of the selected shape to be copied to). The input from the speech matches the expected input from the sketch recognizer. As a result, the system copies the shape located at point 1 to the location at point 2. Figure 4-16 shows the result of this multimodal interaction with the system.

2



Figure 4-15: Two sketching inputs were received during speech input - both inputs were points



Figure 4-16: Result of using multimodal input to "copy" a unit

Chapter 5

Intermediate Feature Recognizer

The Intermediate Feature Recognizer (IFR) is used to recognize shapes of intermediate complexity. Recognizing shapes that have intermediate complexity improves sketch recognition in the COA domain. One purpose of the IFR is shape detection; it is able to recognize certain shapes of intermediate complexity that can't be detected by either the primitive recognizer or the domain recognizer. A second purpose of the IFR is error correction; it can correct errors caused by either incorrect system recognition or the user.

The benefits of the Intermediate Feature Recognizer are summarized below:

- Shape Detection
 - Recognizes variably-drawn shapes of intermediate complexity.
- Error Correction
 - Correct some initial stroke segmentation errors
 - Correct errors dealing with over-tracing and filled-in lines.

5.1 Limitations of LADDER System

Recognition in the LADDER System proceeds in a bottom-up as primitive shapes combine to form domain shapes. Domain shapes can combine with other domain

shapes or primitive shapes to form more complex domain shapes. One problem with the LADDER method of shape definitions is that the description of a shape specifies a specific number and type of components. Shapes with a variable number of components are not able to be recognized, since the LADDER shape description must specify the specific number of components.

The LADDER method of shape definitions is not able to deal with filled-in lines and multi-segment lines. If a user drew a line with one stroke, and then tried to extend the line with another stroke, the system recognizes these strokes as two separate pen-strokes, each of which are classified as a line. The desired system response would be to combine the two lines together to create a single line encompassing the space covered by both lines. While it might appear visually as if a single line was drawn using the pen, using two strokes will create two separate lines that are added to the domain recognizer. Any domain shape that depends on the extended line will not be recognized because the domain shape recognizer is only aware of two lines, each of which are part of the extended line. Therefore, common user errors such as filled-in lines and multi-segment lines won't be recognized by the LADDER domain recognizer.

5.2 Shape Detection using the Intermediate Feature Recognizer

One of the two primary purposes of the Intermediate Feature Recognizer is to be able to detect shapes of intermediate complexity. A shape of intermediate complexity is one that can't be detected by either the primitive recognizer or the domain recognizer. Primitive recognizers process shapes drawn with a single pen-stroke. The domain shape recognizer requires a fixed number of components for each defined domain shape. The intermediate feature recognizer fills the gap in shape detection between the primitive recognizer and the domain recognizer, detecting shapes drawn with more than one stroke as well as detecting variably-drawn shapes.

Intermediate feature recognition is particularly useful to the COA domain because there are several COA domain symbols which can't be described using LADDER shape descriptions. Some examples are dashed frames (which represent pending or suspected location of units) and dashed ellipses (which can be used to represent a seize action.) These features can't be recognized by either the primitive recognizer (because they are drawn with more than one stroke) or the domain recognizer (because they are variably drawn), but they can be recognized using the intermediate feature recognizer.

A LADDER shape description can not be written for a dashed line because the number of dashes can vary. Creating a domain shape description for a variably-drawn shape is not possible since a specific number of component shapes and their type must be specified in LADDER. The intermediate feature recognizer does not have this same requirement - dashed lines are recognized if they contain two or more dashes. There is not a specific number of dashes required for a dashed line. Using the dashed line intermediate feature, LADDER definitions can be written for dashed frames for friendly and enemy units.

5.3 Error Correction using the Intermediate Feature Recognizer

Another purpose of the intermediate feature recognizer is to correct errors in shape recognition. One possible source of error can result from incorrect stroke segmentation by the primitive recognizers. Another source of error is the user, who might make a mistake and then try to correct it. Several examples of user error are filled-in lines and multi-segment lines, both of which can be corrected by the intermediate feature recognizer.

5.3.1 Correcting initial stroke segmentation errors

Incorrect stroke segmentation is a limitation of the current LADDER system. If a user draws a pen-stroke that is recognized as a polyline, the system segments the stroke at each of the segmentation points indicated by the polyline classifier. Then the IFR sends the domain shape recognizer line objects corresponding to the pen stroke between each pair of consecutive segmentation points. Bottom-up recognition occurs using the initial set of line segments. If the initial segmentation of the polyline was wrong, the user must erase the line segments and redraw them.

The IFR analyzes each of the line segments that were created to determine if any of them can be combined to form a single line segment. In order to be combined, line segments must have similar slope and be collinear, with little or no space between the line segments. If it is determined that two line segments may be combined to form a single line, then the new line (extended) is added to the domain recognizer, with the line segments that it is composed of defined as its sub-shapes. In this way, multiple interpretations of the segmentation are produced and added to the domain recognizer.

The domain shape recognizer requires that there is only one final interpretation of the segmentation of a stroke, because each segment is identified by a unique ID, and a unique ID may be recognized only once as part of a domain shape. The final interpretation of the pen stroke may not be known until more shapes are drawn on the screen. If the primitive polyline classifier segments the stroke in too many locations, the final object may not be recognized by the domain recognizer. The IFR solves this problem by combining line segments together if they meet certain geometric constraints. Not only does the initial segmentation get analyzed by the domain shape recognizer, additional segmentations produced by the intermediate feature recognizer are also produced and analyzed. The initial segmentation is produced by primitive recognizers and complemented by segmentations produced by the IFR. The IFR helps to improve domain shape recognition by providing alternate stroke segmentations to the domain shape recognizer.



Figure 5-1: A rectangular frame that contains a gap

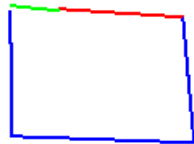


Figure 5-2: The IFR combines the filled-in line (highlighted in green) with the line highlighted in red

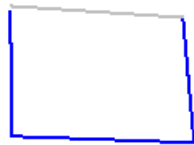


Figure 5-3: The grey line indicates the line recognized by the IFR from the input shown in Figure 5-2

5.3.2 Correcting filled-in lines and multi-segment lines

Each pen stroke produces one or more recognized primitive shapes. Consider a rectangular shape with a gap in the frame, as shown in Figure 5-1. If the user draws a pen-stroke to fill in the gap, a separate primitive line object is created for the filled-in portion. The IFR combines line objects that have a similar slope and are collinear, as it did to correct segmentation errors. The IFR combines the line object for the filled-in section (highlighted in green in Figure 5-2) The IFR creates a new line object (highlighted in grey in Figure 5-3 that encompasses both the original line and the filled-in portion.

In a similar manner, the IFR system can deal with multi-segment lines. Lines that are collinear, have the same slope, and cover the same area can combine to form a new line object that encompasses the over-traced portion and extends to the furthest endpoints of each of its component line segments. This is slightly different than the recognition of filled-in lines, which requires that the lines being combined share one endpoint.

5.4 Recognition of Intermediate Features

5.4.1 Dashed lines

Dashed lines are recognized by the intermediate feature recognizer. Three constraints are used to recognize a dashed-line: slope, collinearity, and coverage. As each new line segment is added to the IFR, it is analyzed to determine if it has a similar slope to other lines and dashed lines that have been recognized so far. For any similar slopes, the IFR checks to see if the new line is collinear with lines of the same slope. The third constraint is coverage of the line segments: This refers to how much the dashes cover the length between the two furthest endpoints of the dashes. If the coverage is small, it could indicate that the lines are far apart. If the coverage is large, it could indicate that the dashes are close together possibly touching. In that case, the lines could combine to form a compound line, described below. If the IFR recognizes line

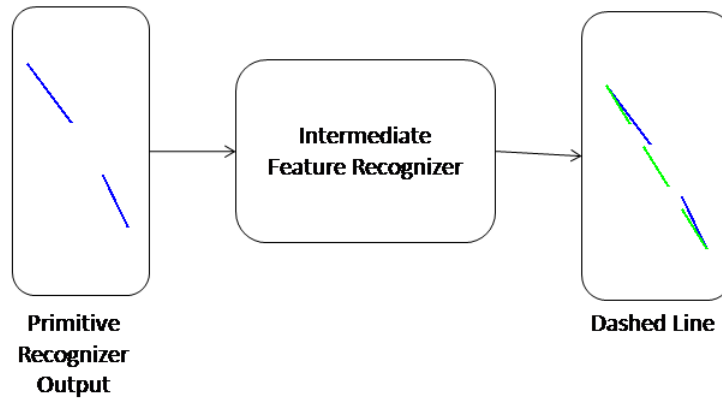


Figure 5-4: A dashed line can be recognized from as few as two dashes

segments as belonging to a dashed line, a dashed line primitive type object is created, with its subcomponents being the line segments (dashes) that it was recognized from.

As each dashed line is recognized, the system displays its interpretation of the intermediate feature on the drawing panel.

5.4.2 Compound lines

Compound lines are also recognized by the intermediate feature recognizer. Compound lines are lines that are drawn with multiple strokes but contain the geometric properties of a single line. Some examples of compound lines are multi-segment lines, where a portion of a line has been drawn-over twice, as well as filled-in lines, where a portion of a line was missing originally and then filled in. The same three constraints that were used to recognize dashed lines are also used to recognize a compound line: slope, collinearity, and coverage. The only difference in recognizing a compound line is that a compound line has a higher coverage than a dashed line does. Once an edge has been recognized, a line object is created and sent to the domain shape recognizer. The resulting line segment produced by the edge represents the line between the two

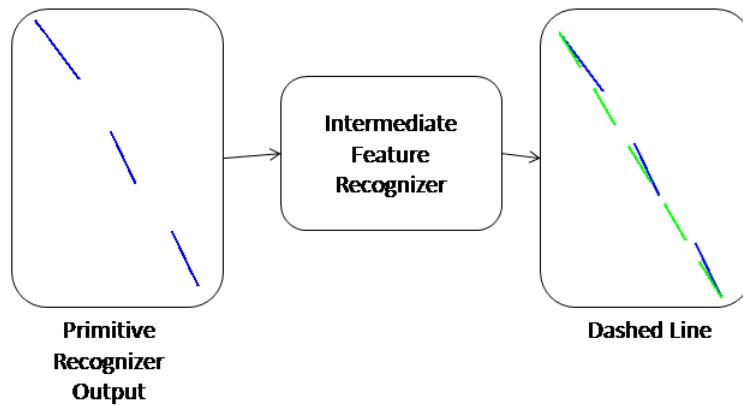


Figure 5-5: As more dashes are recognized as part of a dashed line, the endpoints of the dashed line update

furthest endpoints of the edge, even though the edge might be made with multiple strokes from a multi-segment or filled in line.

5.4.3 Dashed Chains

A dashed chain consists of a sequence of one or more dashed lines connected at their endpoints. It also refers to a single dashed line with zero or more waypoints.

After the IFR recognizes a dashed line, it creates a dashed chain object. As each dashed chain is created, the intermediate feature recognizer analyzes all other dashed chains in the sketch to determine if the new dashed chain can combine with other dashed chains to form a larger dashed chain. If an endpoint of one dashed chain is close enough to the endpoint of another dashed chain, then the dashed chains combine to form a new dashed chain with the common endpoint now a waypoint of the dashed chain.

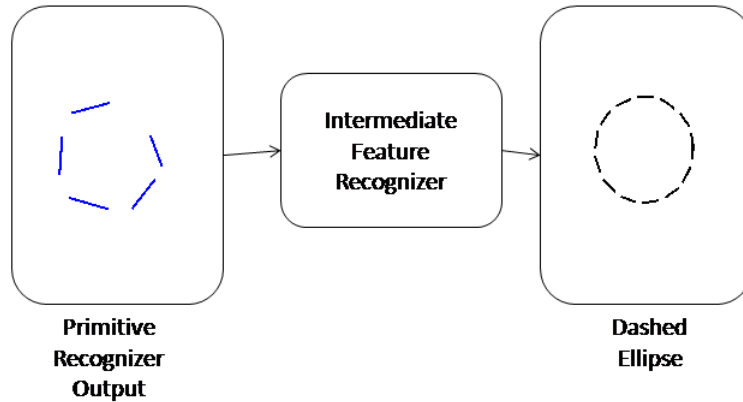


Figure 5-6: The IFR recognizes dashes arranged in a circle as a Dashed Ellipse shape.

5.4.4 Dashed Ellipses

Dashed ellipses are another type of shape recognized by the IFR. In order to recognize a dashed ellipse, the system first detects lines that are drawn near one another, with a similar but slightly differing slope. The method is similar to detection of dashed lines, except that in dashed lines the slope of the dashes should be equal but in dashed ellipses the slope of the dashes should vary slightly. As more dashes are added that fit this pattern, the intermediate feature recognizer extracts the two endpoints and midpoint of each dash for all dashes in the possible ellipse. The resulting collection of points is then sent to the ellipse primitive recognizer, which tries to fit an ellipse to the points. If the primitive recognizer determines that the points form an elliptical shape, then the intermediate feature recognizer creates a dashed ellipse shape.

Chapter 6

Modifications of the LADDER System

The existing LADDER recognition system was able to recognize many shapes in the COA domain. However, the existing system was limited in certain ways, and modifications were made to allow the recognition and editing of more complicated COA domain symbols. This chapter describes some changes made to the existing LADDER system, including the ability to recognize symbols with multiple inheritance and the ability to deal with image replacement and composition in the COA domain.

6.1 Multiple Inheritance

The LADDER System defines and recognizes shapes hierarchically. This system works fine when there is a single way of defining a shape. If there are multiple constructions of a shape, a LADDER shape description can't easily be written that will allow every construction. This is a problem of multiple inheritance.

Consider the example shown in Figure 6-1 which shows the possible ways to construct an enemy mechanized infantry regiment. Starting at the top of the figure, an enemy frame has been drawn. Each arrow leading down indicates one change made to the symbol. There are 20 unique orders that can be used to construct the enemy mechanized infantry regiment, with 16 distinct shapes encountered on the way, each

of which is a valid symbol in the COA domain. Any shape definition that is expressed in terms of a change to the previous unit in the hierarchy would restrict drawing order. This requirement is unnatural to users, so in order to allow symbols to be drawn in any order, context shapes are used.

In the COA domain, modifiers of frames are constructed in the “context” of a frame, and there can be multiple modifiers of frames. The LADDER shape language allows components to be designated as “context” components. However, the system did not treat these differently from other components. In response, the domain recognizer was modified to allow context components to remain in the VSC after they are recognized as a context component of a more complex shape. This allows multiple modifiers to be recognized in the context of a frame. In the COA domain, frames are designated as context shapes. As modifiers are drawn, they too get recognized and are added to the VSC. Defining frames as context components of a shape allows modifiers to be drawn in any order. Using this method of shape definition allows all 20 drawing orders for creating a mechanized enemy infantry regiment. It also allows the recognition of any of the 16 possible shapes that may be drawn in the construction of a the enemy mechanized infantry regiment, as shown in Figure 6-1.

The use of context components make the LADDER shape definition hierarchy compact. Instead of requiring 16 unique LADDER shape definitions for the COA symbols shown in Figure 6-1, these symbols can be recognized using only 7 LADDER shape definitions. Each frame modifier has a unique LADDER shape definition, each of which require the frame as a context component. For the example in Figure 6-1, the 7 required shape definitions are: the three echelon modifiers (company, battalion, and regiment), three icon symbols (infantry, armor, and mechanized infantry), and the enemy unit frame. Using context components in the LADDER shape definitions makes the shape definitions compact.

6.1.1 Symbol Editing

The LADDER system was designed so that each set of pen strokes correspond to one recognized shape. However, with the use of context shapes, one set of pen strokes

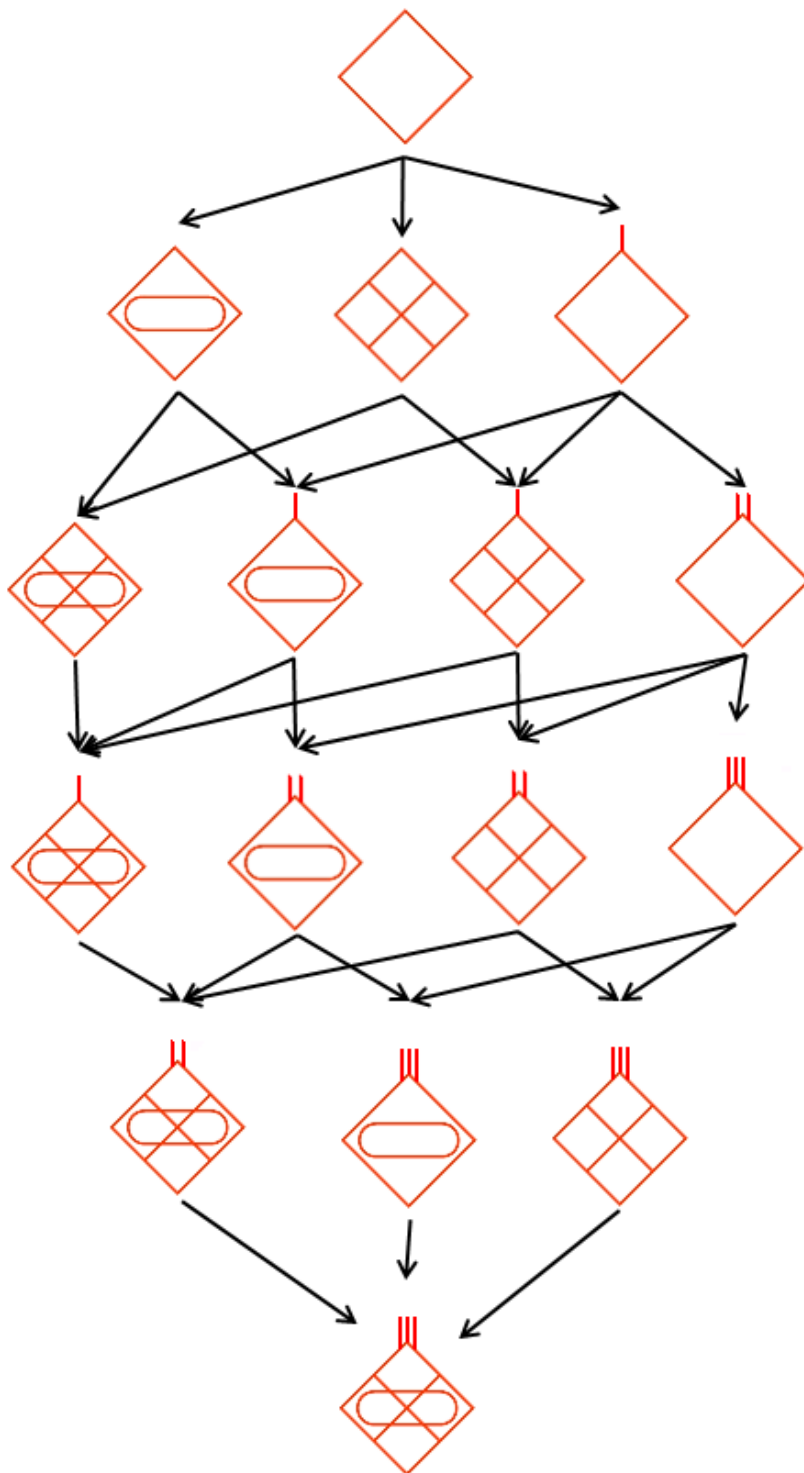


Figure 6-1: Multiple Inheritance of Symbol Construction

may be associated with multiple recognized objects. For example, a set of pen strokes representing a mechanized infantry regiment results in three different shapes in the VSC: the enemy unit frame, a mechanized infantry regiment, and a regiment. From the user's perspective there is a single COA symbol for the mechanized infantry regiment, which is represented by three different shapes in the VSC.

This change in design resulted in the need to change the way recognized shapes are moved and edited, copied, and deleted. Shape modifiers have as a context component the frame that they modify. If a symbol with a modifier is edited (eg. moved), then the frame itself should only be modified once. However, based on the way the system was implemented, moving a symbol resulted in the frame moving twice or three times as much (depending on how many modifiers there were to the frame). There was an extra step needed to make this work with context shapes. For a single edit action on a COA symbol, a collection of shapes in the VSC may be modified. The LADDER domain recognizer was modified to keep track of which components were modified in a single edit action, and did not process duplicate edit commands for a given component. This modification fixed the move, copy, and delete commands errors associated with the original implementation of the system.

6.2 Image Replacement and Composition

The display component of a LADDER shape definition is used to specify how each shape should be displayed. This works for separate shapes, but it doesn't work for the compositional nature of COA shapes. As was discussed in Section 6.1, there is a unique recognized shape for each frame and for each modifier of that frame. Just as modifiers of an image are recognized separately, the respective images for each modifier are displayed separately.

As each image replaces pen strokes shown on the screen, the visual representation changes, but the underlying stroke location data does not. The system still uses the underlying location data of pen strokes for recognition, but if the image is not aligned with the original strokes, new strokes drawn over the image will not match with the

old strokes. The problem is due to the possible difference in the scaling, translation, or rotation between the displayed image and the original sketched strokes. Since shapes are recognized hierarchically, users may draw over an image and expect a more complicated shape to be recognized. However, if the original pen strokes are not of the same scale as the displayed image, the complex shape will not be recognized.

Figure 6-2 is a visual example of the problem that would arise if the image doesn't match up with the stroke data. Figure 6-2a shows pen strokes the user has drawn to indicate a friendly infantry unit. After the friendly unit frame is drawn, it is replaced by its associated image, as shown in Figure 6-2b. The system would not recognize a friendly infantry unit even though the geometric properties of a friendly infantry unit appear to be met from the user's perspective. Figure 6-2a shows the pen input to the system, which shows that the pen strokes do not satisfy the geometric properties of a friendly infantry unit because the endpoints of the two diagonal lines do not touch the corners of the frame. If the LADDER shape definition is used to display an image when a shape is recognized, the stroke data may not match with the displayed image, which may prevent recognition of more complex shapes (as shown in Figure 6-2).

6.2.1 Preserving Scale, Translation, and Rotation

In order to correct the situation where the image differs from underlying shape data, the components of the shape must be modified to fit the location of those components in the image. The requirement for modification is to reset the location of each component of the shape to its correct location in the image, based on the scale, translation and rotation of the image. However, the image itself doesn't contain any information about components which can be used to set the location of the shape components.

One solution to this problem is to use the LADDER Shape Definition itself to encode how components of a shape should be modified when replaced by an image. The idea is to pair each name of a shape component in a shape definition with a unique corresponding scale, translation, and rotation transform which is used to determine how the component itself is modified and how the shape is displayed. The shape definition itself encodes:

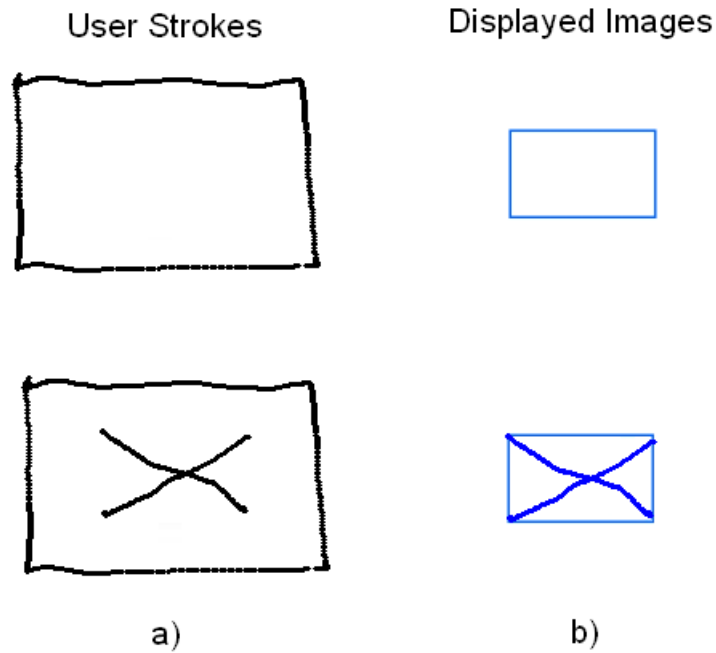


Figure 6-2: The geometric properties of displayed images may not match with input pen strokes

- how components of a shape should be modified when replaced by an image of that shape.
- how the image should be scaled, rotated, and translated when displayed.

Preserving Translation

The name of each component of a shape is used to determine the new location of the component in relation to the displayed image for the shape. The geometric properties of a shape are used to recognize the shape, independent of scale. The new location of the components of a shape is based on the name of the component as defined by the shape definition for the recognized shape.

Each of corner of an image has a unique name, as does the midpoint and each side of a shape. For the modifiers recognized by the system, this set of points is sufficient to enable hierarchical recognition. As other modifiers and more complex shapes are added additional reference points may need to be added. In addition to points, some

sides and diagonals are also referenced. This collection of points/lines is used to set the correct location of a component/alias based on the image scale, rotation, and translation.

Preserving Scale

In a COA diagram, some symbols are displayed at a constant scale, while others vary according to how the shape is drawn. Frames and frame modifiers are examples of shapes displayed at a constant scale, while minefields and actions are examples of shapes displayed at the size they were drawn. In order to preserve scale for both image display and component relocation, a special designator for preserving scale (“PS”) is used to indicate that scale should be preserved for the shape as well as to indicate which component should be used to calculate scale ratio for resizing the image.

For example, Figure 6-3a shows two sketched rectangles. After these strokes are recognized as a friendly unit, fixed-size images are displayed, as shown in Figure 6-3b. Scale is not preserved for unit frames. However, scale is preserved for minefields. Figure 6-4a shows the stroke input for two minefields. The images that the system displays are shown in Figure 6-4b. Each image is scaled to the size of the stroke input for the minefield, because scale is preserved for minefield symbols.

Preserving Rotation

The rotation of images and symbols varies in a Course of Action diagram. Units and unit modifiers are not rotated; they are always displayed at a canonical orientation. Actions and minefields are examples of COA symbols that should be displayed at the angles drawn. In order to indicate which shapes should preserve rotation, the designation “PR” is added after the name of one component in a shape definition. This designation is used to indicate that rotation should be preserved for the shape.

In order to detect rotation in a shape, each shape definition contains two designated reference points that are used to calculate its rotation. For shapes missing these references, it is assumed that rotation is not preserved. If the references do exist, its

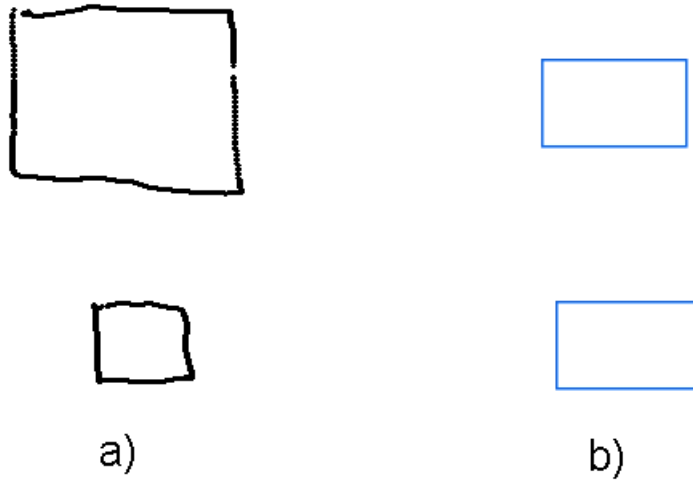


Figure 6-3: User strokes indicating a friendly unit are replaced by a fixed-size image

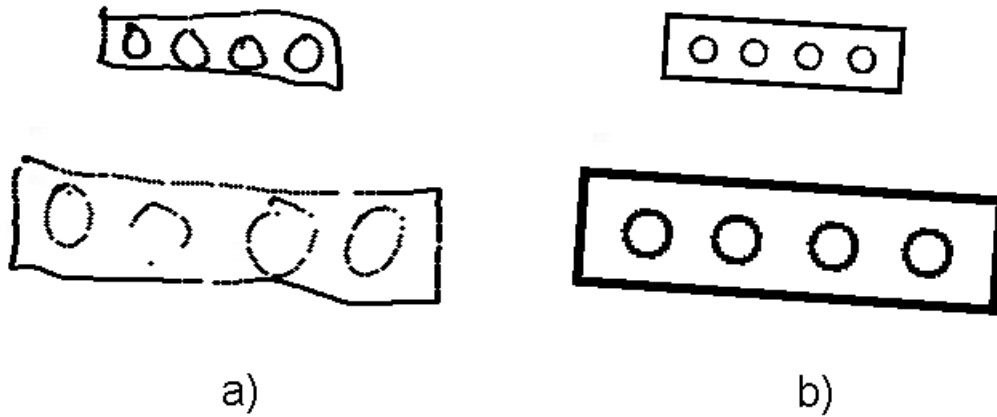


Figure 6-4: User strokes indicating a minefield are replaced with an image scaled to the size of the sketched minefield

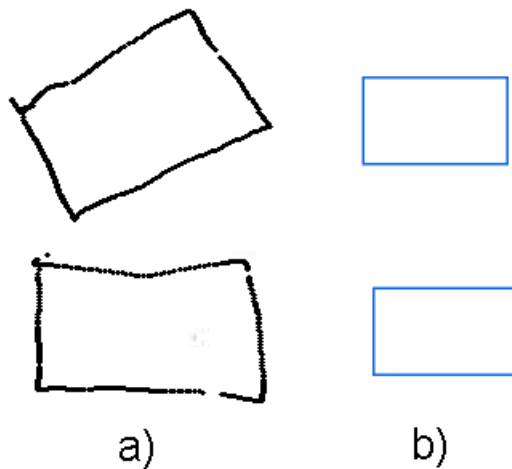


Figure 6-5: User strokes indicating a friendly unit are replaced by a horizontal image, regardless of the orientation of the sketched rectangle

orientation is computed as the angle between this pair of points and the horizontal. Shapes missing these reference points are always placed in a standard orientation.

Figure 6-5a shows two rectangles that have been drawn at different orientations. These friendly unit frames are displayed as images at a fixed orientation (the horizontal - as shown in Figure 6-5b) because rotation is not preserved for unit frames. Rotation is preserved for minefield symbols. Figure 6-6a shows two minefields that have been sketched at different orientations. The images the system displays for the recognized minefields are shown in Figure 6-6b. The minefield images were rotated to match the orientation of the sketched strokes because rotation is preserved for minefield symbols.

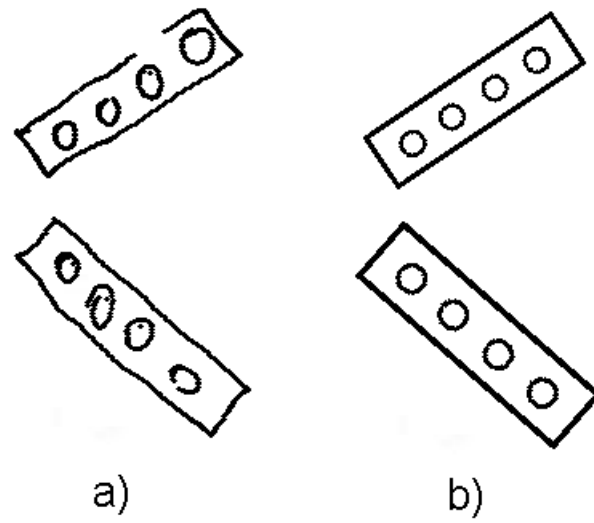


Figure 6-6: User strokes indicating a minefield are replaced with an image at the same orientation as the sketched strokes

Chapter 7

Results

The Course of Action Design Interface developed for this project demonstrates a successful system capable of recognizing a Course of Action sketch. It is interactive and allows the user to get immediate feedback when a symbol is recognized. It is capable of transforming the input from sketching shown in Figure 7-1 to the automatically generated output shown in Figure 7-2.

7.1 Recognized Shapes

The COA Design Interface is capable of recognizing a number of symbols in the COA domain, including unit frames, frame modifiers, action symbols, and other COA object symbols.

Table 7.3 displays the two frames - rectangular (friendly) and diamond (hostile) - that can be recognized. Echelon and icon modifiers can be added to each frame. Table 7.4 shows echelon modifiers that can be drawn on friendly unit frames, while Table 7.5 shows echelon modifiers that can be drawn on enemy unit frames. Icon modifiers can also be added to each of the frames. Icon symbols can be used for both friendly and enemy units (depicted in Tables 7.6 and 7.9 respectively). Icon modifiers can also be represented as a task organized symbol for friendly units only, and are shown in Table 7.8.

Action symbols can also be recognized using the software system. Table 7.10

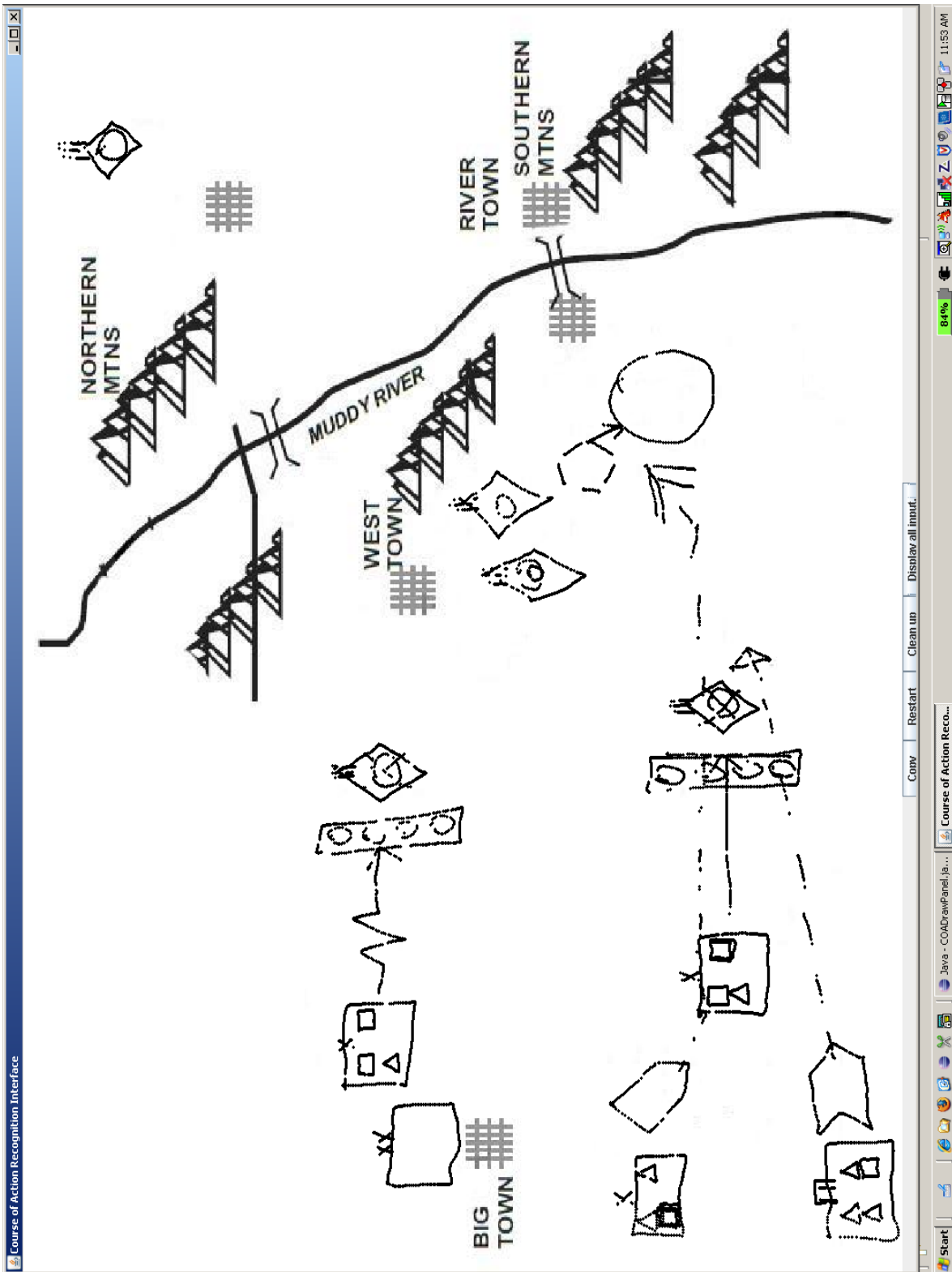


Figure 7-1: Sketched Input to COA Design Interface

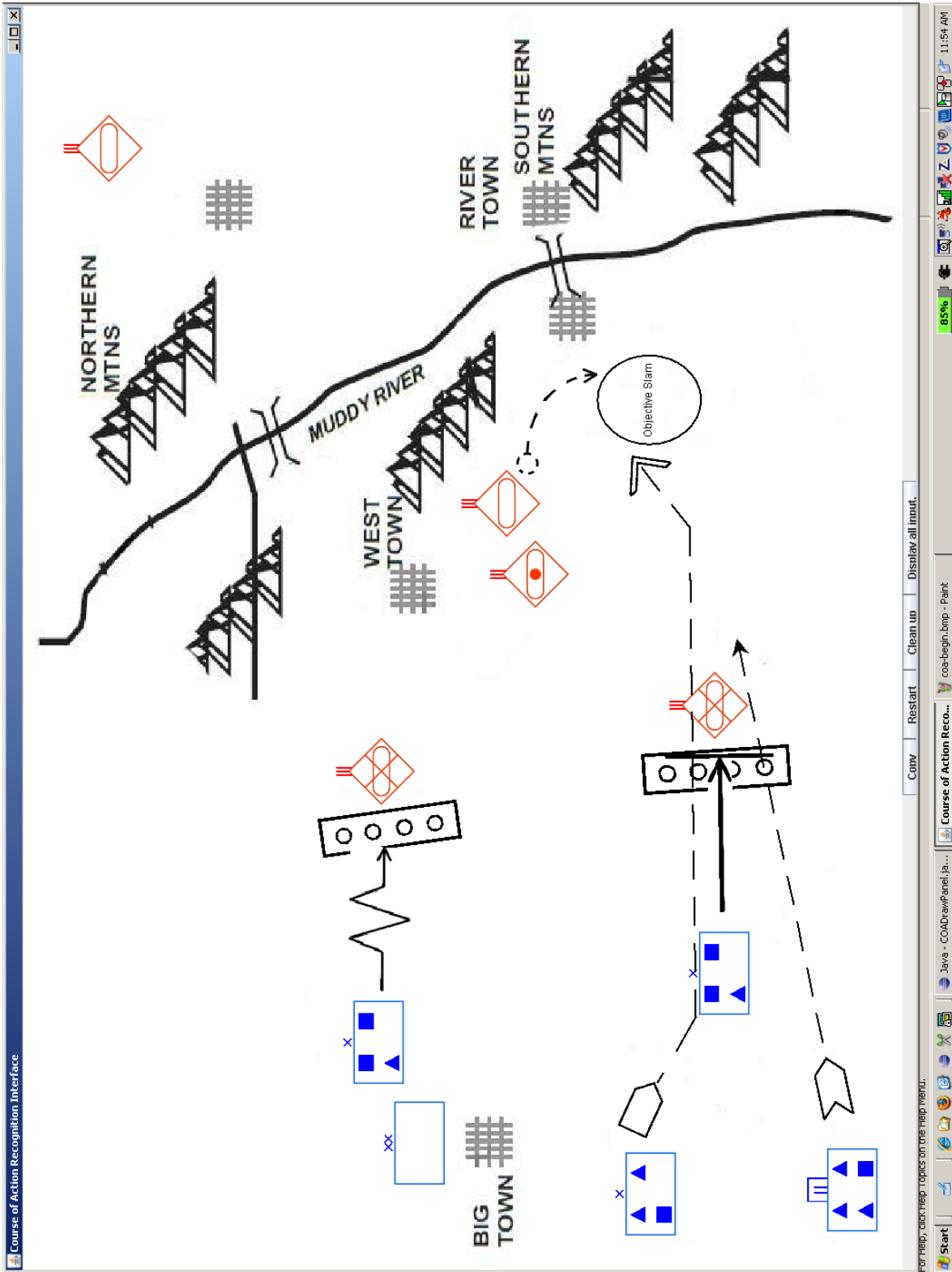


Figure 7-2: Displayed Output from COA Design Interface

displays examples of recognized action symbols. The system also recognizes other COA symbols shown in Table 7.11.

7.2 Combinatorics

The shapes depicted in these tables are only a fraction of the total number of shapes that the software system can recognize. Due to the compositionality of COA unit frames, different frame modifiers can be combined for the same unit. For friendly units, this results in 180 possible units (10 friendly echelon modifiers (including unknown) * 13 friendly icon symbols (including no icon) + 10 enemy echelon modifiers * 5 enemy icon symbols. This does not include possible task organization symbols. The number of task organized combinations possible is shown in Table 7.1. If combined with echelon modifiers, the task-organized modifier allows the recognition of an additional 140 units (10 echelon modifiers * 14 task-organized symbols). In addition to units, there can be five action symbols and two additional object symbols (minefield and planned minefield). In total, the COA Design Interface is capable of recognizing 247 symbols, summarized in Table 7.2.

Number of Task-Organized Symbols in Frame	Number of Combinations
1	2
2	3
3	4
4	5
Total:	14

Table 7.1: Task-Organized Icon Modifier Combinations

7.3 Future Work

7.3.1 Future Capabilities of Multimodal Input

Multimodal interaction can give meaning that is not clearly expressed through only one modality, and speech input can be used to provide top-down recognition of shapes.

Type of Symbol	Number Recognized
Unit symbols (w/ echelon and icon symbol modifier)	180
Unit symbols (w/ echelon and icon task-composition modifier)	140
Action Symbols	5
Object Symbols	2
Total:	327

Table 7.2: Summary of Recognized Shapes



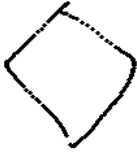

Symbol Description	Input pen strokes	Displayed Image
Friendly Ground Unit		
Enemy Ground Unit		

Table 7.3: Recognized COA Frames

The ultimate goal of future extensions to the system is to allow the user to interact with the system in a natural and easy manner, while improving recognition performance.

Shape Creation

Recognition using the LADDER recognition system uses bottom-up recognition to recognize complex shapes. Complex shapes are not recognized correctly if any of the primitive or intermediate shapes that it is composed of are not recognized correctly. In order to facilitate complex object recognition, a user could simultaneously state the class of the object when drawing the object. In this manner, the speech input could be used to allow top-down recognition of pen strokes. The user may incorrectly draw or state the classification of the shape, so it will be important to determine how the input from the two modalities should be combined.







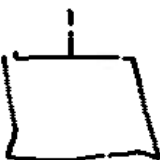

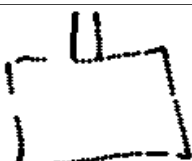



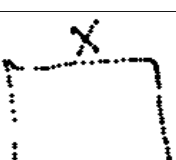
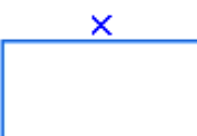
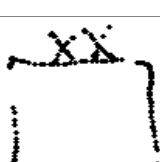
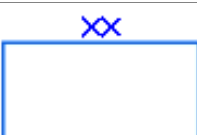

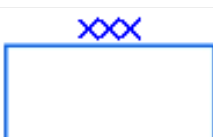
Symbol Description	Input pen strokes	Displayed Image
Friendly Squad		
Friendly Section		
Friendly Platoon		
Friendly Company		
Friendly Battalion		
Friendly Battalion Task Force		
Friendly Brigade		
Friendly Division		
Friendly Corps		

Table 7.4: Recognized COA Echelon Modifiers of Friendly Units










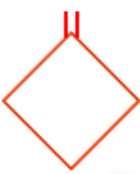

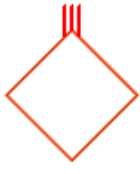




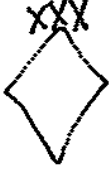

Symbol Description	Input pen strokes	Displayed Image
Enemy Squad		
Enemy Section		
Enemy Platoon		
Enemy Company		
Enemy Battalion		
Enemy Regiment		
Enemy Brigade		
Enemy Division		
Enemy Corps		

Table 7.5: Recognized COA Echelon Modifiers of Enemy Units










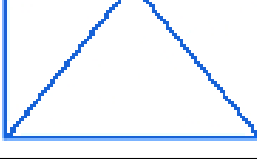

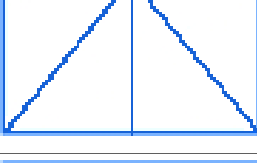

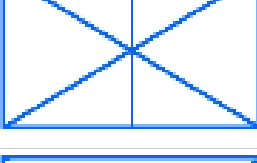
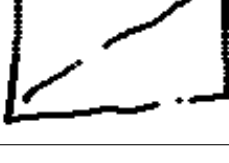
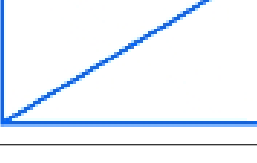
Symbol Description	Input pen strokes	Displayed Image
Friendly Infantry		
Friendly Armor		
Friendly Mechanized Infantry		
Friendly Self-Propelled Howitzer/Gun		
Friendly Anti-Armor		
Friendly Motorized Anti-Armor		
Friendly Motorized Infantry		
Friendly Recon		

Table 7.6: Recognized COA Icon Symbol Modifiers of Friendly Units - 1


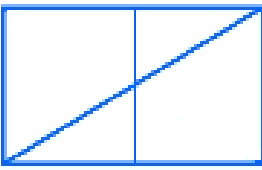

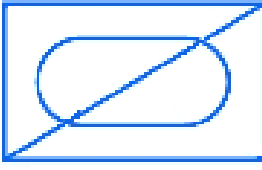



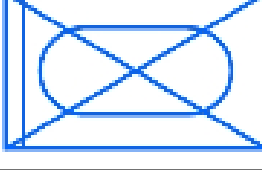
Symbol Description	Input pen strokes	Displayed Image
Friendly Motorized Recon		
Friendly Armored Recon		
Friendly Armored Anti-Armor		
Friendly Mechanized Infantry w/ Gun Systems		

Table 7.7: Recognized COA Icon Symbol Modifiers of Friendly Units - 2



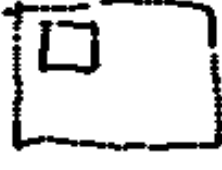

Symbol Description	Input pen strokes	Displayed Image
Friendly Armor (Task Organized)		
Friendly Mechanized Infantry (Task Organized)		

Table 7.8: Recognized COA Icon Task Organized Modifiers of Friendly Units









Symbol Description	Input pen strokes	Displayed Image
Enemy Infantry		
Enemy Armor		
Enemy Mechanized Infantry		
Enemy Self-Propelled Howitzer/Gun		

Table 7.9: Recognized COA Icon Symbol Modifiers of Enemy Units



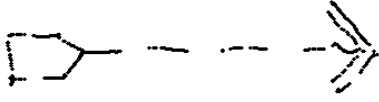
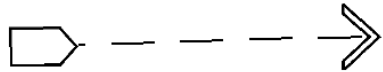

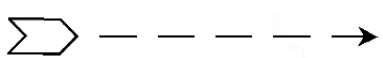
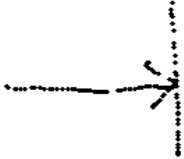
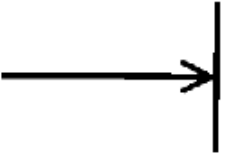
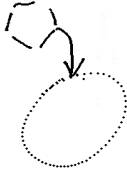
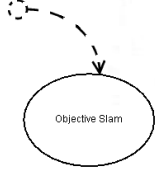
Symbol Description	Input pen strokes	Displayed Image
Fix Action		
Follow and Assume Action		
Follow and Support Action		
Penetrate Action		
Seize Action		

Table 7.10: Recognized COA Action Symbols


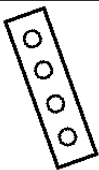
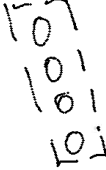

Symbol Description	Input pen strokes	Displayed Image
Minefield		
Minefield (Planned)		

Table 7.11: Recognized COA Object Symbols

Shape Naming

In the Course of Action domain, it may be useful to be able to name sketched shape. Examples of shapes that may be named in the CoA domain are objective areas, phase lines, and units. Pen-input recognition becomes much more complex when mixing shape recognition and character recognition in the same sketch. Since names are often indicated in the textual description, adding text recognition might not be useful. Instead, speech input could be used to input the names of symbols. Objective areas and detailed information about the specific military unit may be indicated by speech.

Assigning Relations

There are some kinds of information that are not easily communicated with pen strokes. This might be as a result of the pen stroke being difficult to draw/recognize or because there is no gesture capable of expressing that information. One example is the task of assigning targets to artillery units in a CoA diagram. There isn't a symbol used to represent this; it is specified in the textual attachment to the CoA diagram. A speech interface could be used to communicate this information.

Intent

Course of Action diagrams are commonly accompanied by a written textual description. While the diagram indicates which units are present and which actions they are to perform, the additional written description is used to explain the order of events and reasons why an action is to be performed. These components indicate the commander's intent for a possible battle, and for this reason the text is an integral component of the operational picture that can't be represented by graphical symbols. Although commander's intent can't be represented by symbols, it is still an essential component of the operational picture.

Chapter 8

Conclusion

I developed a software system that interprets a hand-drawn Course of Action diagram. The system is capable of recognition of objects in the informal sketch, which can then be copied, moved, and deleted. The system understands details about Course of Action symbol geometry, encoded in a collection of LADDER shape definitions. Several modifications were made to the existing LADDER system - allowing (1) multiple inheritance of domain shapes, and (2) replacing informal object with an image while still being able to preserve the scale, rotation, and translation of components of the object. Several system extensions were also developed, including the Intermediate Feature Recognizer, COA Domain Handler, and Multimodal Recognizer. The Intermediate Feature Recognizer is capable of recognizing shapes of intermediate complexity as well as being able to automatically correct some stroke segmentation errors, as well as deal with filled-in and multi-segment lines. The COA Domain Handler is capable of interpreting the sketch, to get some level of understanding of the relations between units and actions. The COA Domain Handler also applies a template grammar, which allows the recognizer to only recognize combinations of COA symbols which are valid in the COA domain. Some types of information are not easily communicated through sketching, so the Multimodal Recognizer is used to combine speech and sketching input. The system is capable of recognizing 327 common Course of Action symbols. The system allows the user to create sketches in a natural manner, through sketching, but the system understands the sketch - creating a formal Course

of Action diagram from an informal sketch.

Bibliography

- [1] Christine Alvarado and Randall Davis. Preserving the freedom of sketching to create a natural computer-based sketch tool. In *Human Computer Interaction International Proceedings*, 2001.
- [2] Christine Alvarado and Randall Davis. Resolving ambiguities to create a natural sketch based interface. In *Proceedings. of IJCAI-2001*, August 2001.
- [3] Randall Davis. Sketch understanding in design: Overview of work at the MIT AI lab. *Sketch Understanding, Papers from the 2002 AAAI Spring Symposium*, pages 24–31, 2002.
- [4] Department of the Army, Washington, DC. *Operational Terms and Graphics*, no. fm 1-02 edition, September 2004.
- [5] K. Forbus, J. Usher, and V. Chapman. Sketching for military courses of action diagrams. In *Proc. IUI '03*, Miami, Florida, January 2003.
- [6] Tracy Hammond and Randall Davis. LADDER: A language to describe drawing, display, and editing in sketch recognition. In *Proceedings of the 2003 International Joint Conference on Artificial Intelligence (IJCAI)*, pages 461–467, Acapulco, Mexico, 2003.
- [7] Tevfik Metin Sezgin, Thomas Stahovich, and Randall Davis. Sketch based interfaces: Early processing for sketch understanding. In *Workshop on Perceptive User Interfaces, Orlando FL*, 2001.
- [8] Shimon Ullman, Michel Vidal-Naquet, and Erez Sali. Visual features of intermediate complexity and their use in classification. *Nature Neuroscience*, pages 682–687, July 2002.
- [9] V. Zue, J. Glass, D. Goodine, M. Phillips, and S. Seneff. The summit speech recognition system: Phonological modelling and lexical access. In *Proc. Intl. Conf. on Acoustics, Speech, and Signal Processing*, pages 49–52, Albuquerque, July 1990.