

Commander's Course of Action Tool (CCOAT)

Final Report

Prepared by
Draper Laboratory
and
MIT Computer Science and Artificial Intelligence Lab

Submitted to
Colonel John "Buck" Surdu
DARPA IPTO

October 2007



Table of Contents

1	Introduction.....	3
2	Example Battle Scenario.....	5
3	CCOAT Prototype Description.....	7
3.1	Graphical User Interface.....	7
3.2	Usage.....	8
4	Software Architecture.....	11
4.1	Sketch and Speech Recognition.....	12
4.1.1	The Sketch Recognizer.....	12
4.1.2	COA Domain Handler.....	15
4.1.3	Multimodal Recognizer.....	16
4.1.4	Tactical Actions.....	18
4.2	Core.....	19
4.2.1	Data Models.....	19
4.2.2	Architecture.....	21
4.3	Graphical User Interface.....	24
4.3.1	Gantt Chart Panel.....	25
4.3.2	Other Panels.....	27
5	Hardware Architecture.....	28
6	Simulating Courses of Action.....	29
6.1	Execution Flow.....	29
6.2	Data Structures.....	31
6.3	Activity Types and Behaviors.....	32
6.4	Animation.....	35
6.5	Monte Carlo Evaluation.....	36
7	References.....	40

1 Introduction

The Tactics Seedling project produced a prototype application tool, called CCOAT (Commander's Course Of Action Tool). The tool demonstrates how Courses of Action (COA) can be sketched free-hand onto a tablet computer, understood, and simulated to verify intent and perform after-action evaluations. The essential steps in a complete COA development cycle are shown in Figure 1-1, where the user first sketches, using a stylus, the desired COA (A). As the sketch is drawn, recognition and encoding software interpret the sketched symbols and steadily builds up an internal representation of the COA (B). This representation is of a form that can be input directly to simulation software for execution (C), enabling subsequent analysis and assessment, if desired (D). All except this last step, assessment, was accomplished within the scope of the Seedling project.

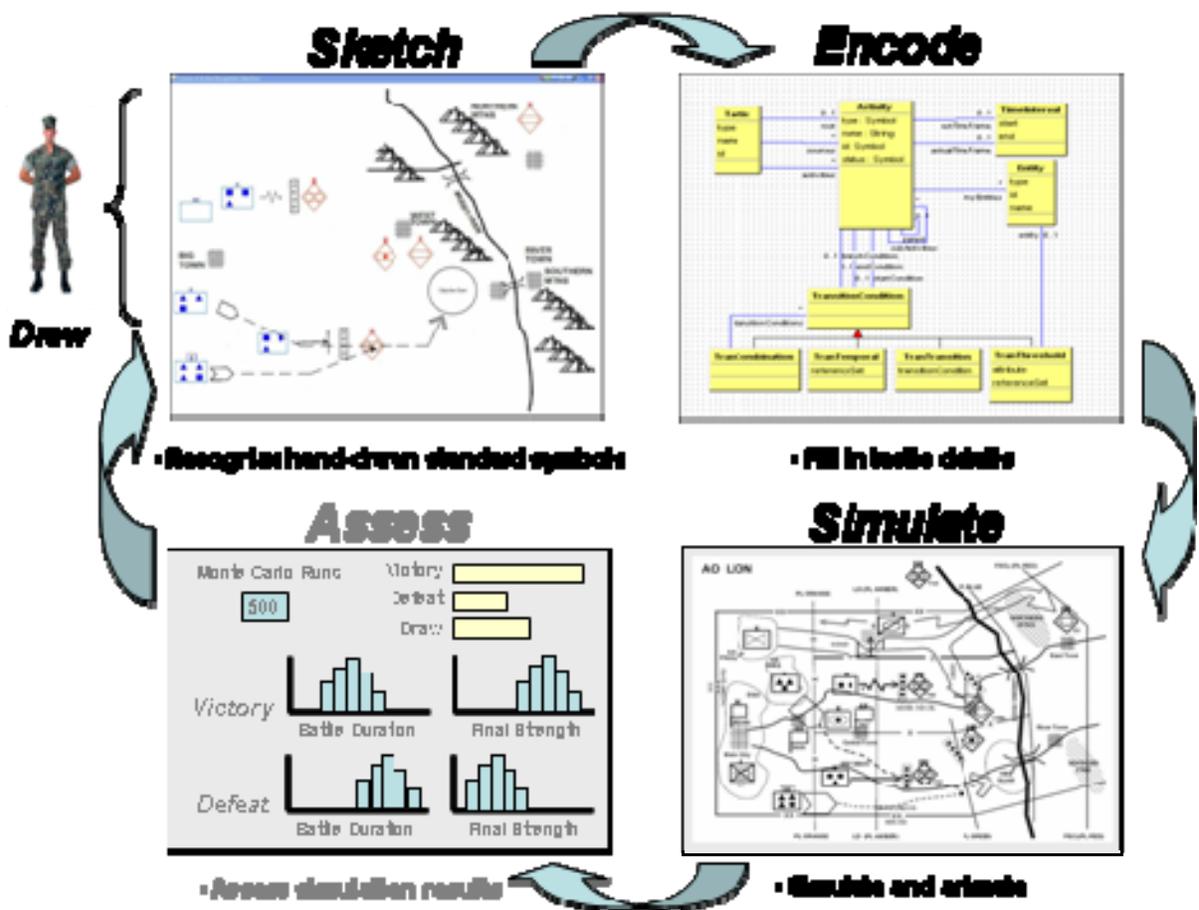


Figure 1-1: Course of Action Development Cycle

Though beyond the scope of the Seedling project, the intent of CCOAT's design was to allow useful extensions that can support additional functions of interest, such as:

- **Tactics Learning** – Courses of Action, or tactics, sketched in by the user are easily captured and stored. An additional indexing step that characterizes the context, within which a particular tactic is produced, would allow them to be recalled in similar situations and at a later time, by other users of CCOAT.

- **Tactics Evaluation and Comparison** – The simulation capability of CCOAT allows tactics to be played out in Monte Carlo manner, under various conditions, to yield basic performance measures, such as duration of operation, victory/defeat/draw, efficiency, etc. Enhancing CCOAT with AI technology would allow the sketched tactic to be evaluated against accepted practice, doctrine, and other sources of tactical expertise.
- **Tactics Tutor** - A library of good tactics, over a range of battlefield situations, coupled with the ability to evaluate and compare them with newly sketched tactics are the two main elements needed for a computer-based Tactics Tutor.

This document will begin in Section 2 with a description of the battlefield scenario, and associated Course of Action, that was used help guide the design and development of CCOAT. Then, in Section 3, a high-level description of CCOAT and its usage is provided. Following this, in Sections 4 and 5, more detailed architectural details are described, for both the software and hardware on which CCOAT runs. Finally, a brief description of the simulation used to execute a sketched COA is provided.

A movie animation that demonstrates how CCOAT is used accompanies this document as a separate “appendix” (file: CCOAT Demonstration Oct07.wmv).

2 Example Battle Scenario

CCOAT is designed, fundamentally, to interpret the sketch of any Course of Action, though its current vocabulary of MIL STD 2525b symbols is small. The battlefield scenario used to help guide the design and scope of CCOAT, and which can be demonstrated by the tool, was adapted from an example in FM101-5. It was reduced in scope (described below) for the seedling project, but for overall context, the larger, division-seized example is described here and depicted in Figure 2-1.

Objective

A mechanized division is to attack and seize OBJ SLAM, at 1500 Apr 12, in order to protect the northern flank of the main effort.

Elements of Scenario

- 1) A mechanized brigade (A) attacks in the north, as an economy of force, to Fix enemy forces, (a), in a zone denying them the ability to interfere with the main effort's attack in the south.
- 2) A mechanized brigade (B) in the south attacks to Penetrate enemy forces in the vicinity of PL AMBER to create sufficient maneuver space to allow the main effort to pass to the east, without interference from the defending enemy infantry regiment (b).
- 3) A tank-heavy brigade (C), the main effort, passes through the southern mechanized brigade (B) and attacks to seize the terrain vicinity of OBJ SLAM, denying the enemy access to the terrain south and west of RIVER TOWN.
- 4) The division reserve (D), a tank task force, initially follows the southern mechanized brigade (B) prepared to contain enemy forces capable of threatening the main effort's passage, then, if not committed west of PL GREEN, follows the main effort prepared to block enemy forces capable of threatening its movement west, ensuring the seizure of OBJ SLAM.

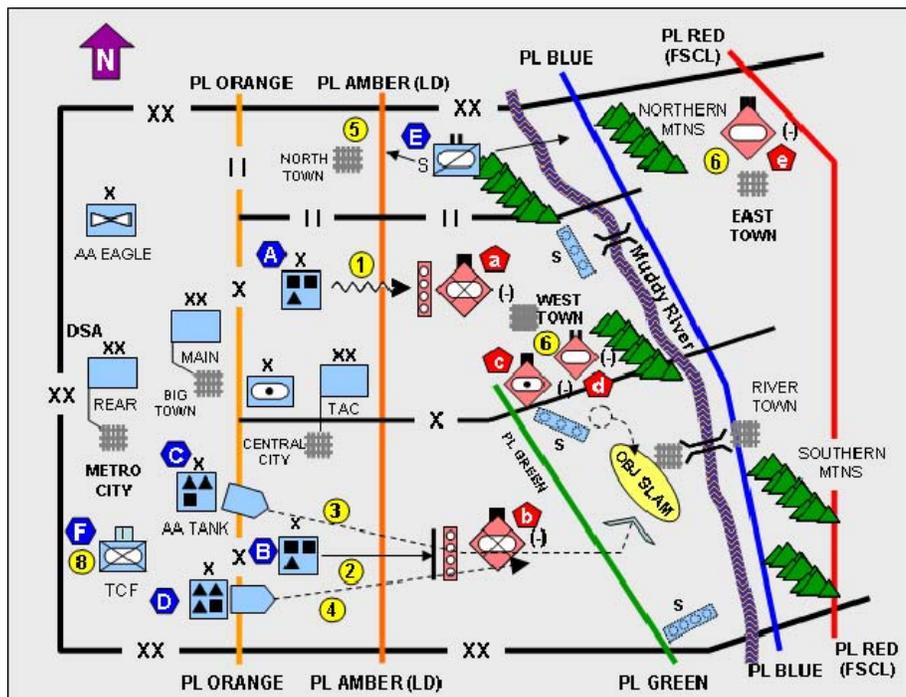


Figure 2-1: Example Battle Scenario from FM101-5

- 5) The divisional cavalry squadron (E) screens the division's northern flank to provide early warning of any enemy force capable of threatening the division's northern mechanized brigade.
- 6) Division deep operations will:
 - 1. Initially attrit enemy artillery, (c), capable of ranging the point of penetration to prevent it from massing fires against the two southern brigades;
 - 2. Then interdict the enemy tank regiment, (d), south of WEST TOWN to prevent its movement south and west towards the main effort.
 - 3. Interdict the enemy tank regiment, (e), north of EAST TOWN to prevent its movement west of the PL BLUE, allowing the main effort sufficient time to seize OBJ SLAM.
- 7) Division fires will:
 - 1. Isolate the point of penetration, allowing the southern mechanized brigade (B) to conduct a penetration.
 - 2. Prevent enemy artillery from massing fires against the two southern brigades.
 - 3. Support deep operations to prevent uncommitted enemy forces from interfering with the initial penetration or the seizure of OBJ SLAM.
- 8) A mechanized infantry team (F) acts as the division TCF with priority of responding to any Level III threat to the division's Class III supply point in the vicinity METRO CITY to ensure the uninterrupted flow of Class III.

Reduced Scenario

The larger was reduced in scope to better align with the allocated resources and the project's period of performance. It is shown in Figure 2-2 and only includes the principal actors, and the first four activities described above.

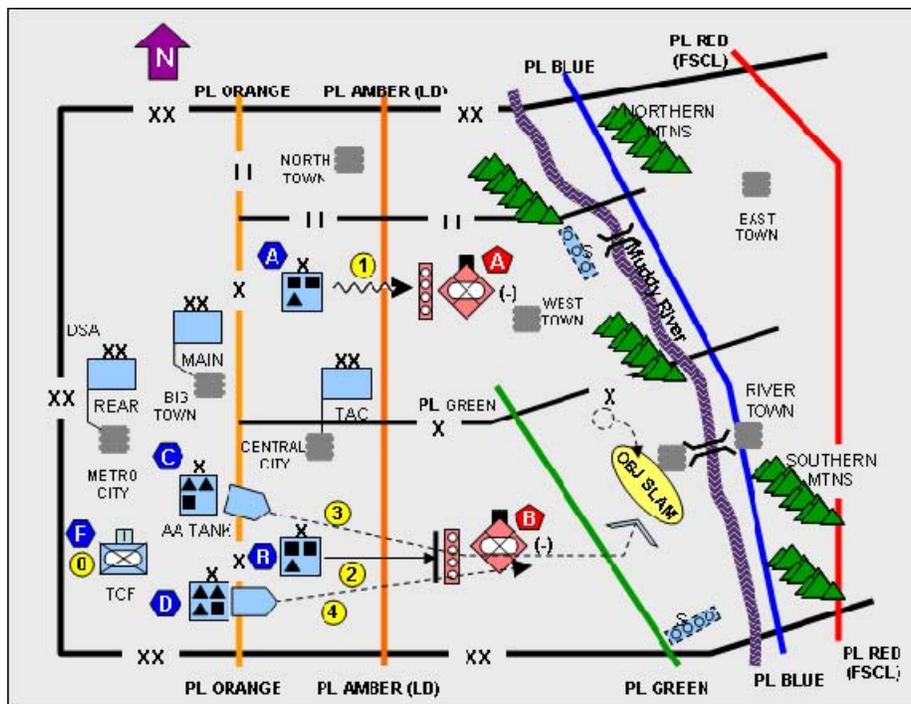


Figure 2-2: Reduced Scenario

3 CCOAT Prototype Description

A description of CCOAT, from the user's point of view, is provided in this section, starting with the user interface, followed by a brief section on its usage.

3.1 Graphical User Interface

CCOAT is a prototype application that allows users to sketch plans and then watch them in action through simulation. The user interacts with the application through the graphical interface shown in Figure 3-1, which is comprised of four panes:

- **Sketch Pad** – With a stylus, the user sketches standard MIL STD 2525b icons on a touch sensitive tablet computer. Once recognized, the user's pen strokes are cleaned up, scaled appropriately, and colored to yield neat diagrams of the intended Course of Action. More about the sketch pad can be found in the discussion of Magic Paper (Section 4.1).
- **Scenario Data View** – The tabbed notebook widget in the GUI can display (or elicit) a variety of information associated with the entities and activities being sketched. For the Seedling prototype, CCOAT's understanding of what is being sketched is displayed back to the user for validation.

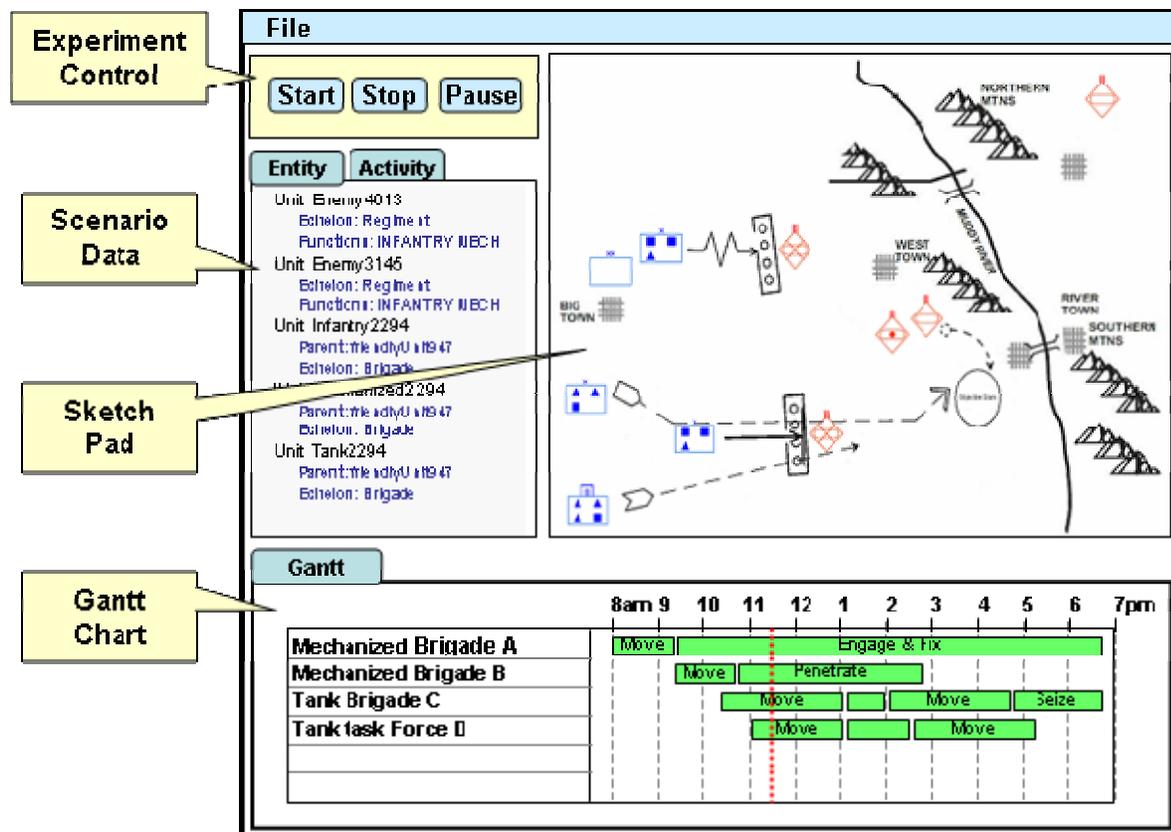


Figure 3-1: CCOAT Graphic User Interface

- **Gantt Chart** – Activities and their timing are picked up and displayed by the Gantt chart, as they are drawn on the sketch pad. With respect to timing and inter-activity temporal relationships, the Gantt chart will do its best to automatically reflect the user's intent.

Usually, not enough information can be picked up in the sketch to be certain the user's temporal intent is honored. In these cases, the user adjusts the relative position of activities in the Gantt chart to more accurately capture timing information. More about how the user can manipulate activity interrelationships and timings is described in Section 4.4, Graphical User Interface.

- **Experiment Controls Area** – The buttons in this area allow the user to start, stop, and pause the simulation of the sketched Course of Action.

3.2 Usage

The user develops a Course of Action principally by drawing it out in the sketch panel, but also through manipulations that can be performed on the Gantt chart. A few steps in this process are captured in the progressive snapshots shown in Figures 3-2 through 3-4.

Compound Unit Sketched

This window shows a single, compound unit having been drawn. The scenario data pane displays the incremental understanding CCOAT has, as the individual elements of the unit symbol are sketched. The order in which these elements are drawn is immaterial, and it is the very last strokes that designate the unit's echelon, in this case. The Gantt chart reflects the existence of the newly sketched mechanized brigade, though no activities for it are indicated, as none have yet been drawn.

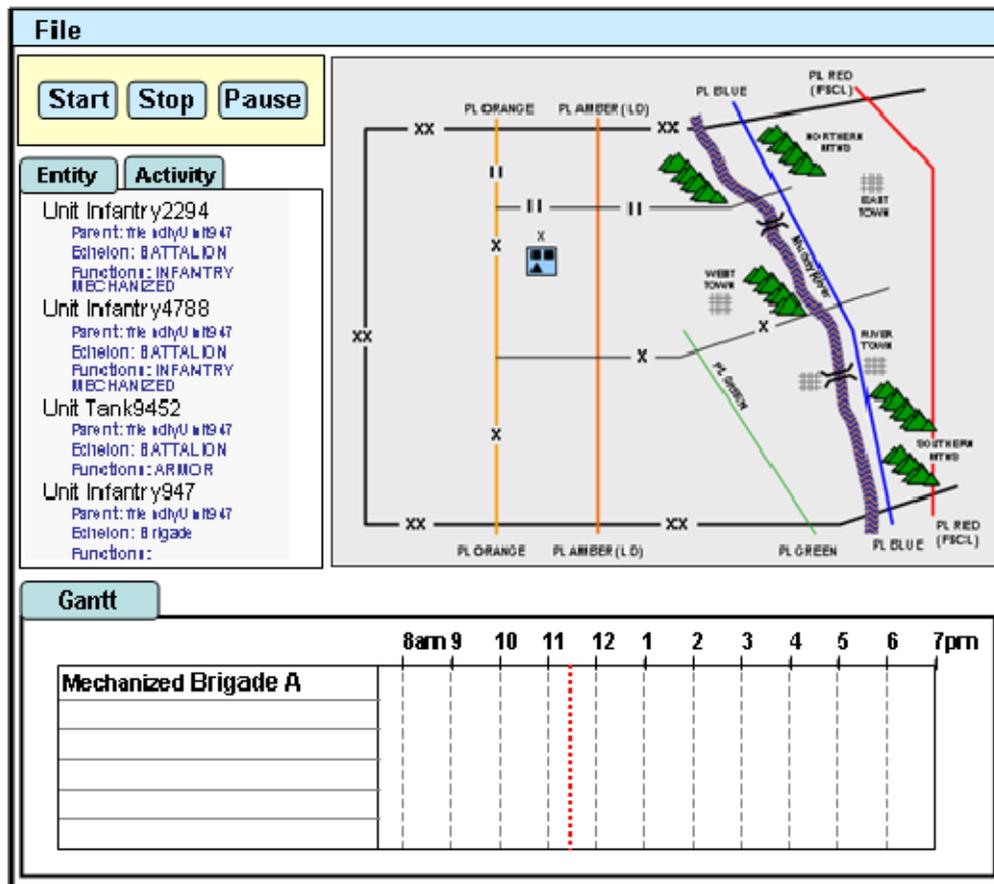


Figure 3-2: Single Compound Unit

“Fix” Operation Sketched

This window shows the sketch of three additional elements: an enemy regiment, a protective minefield, and a Fix operation. Once again, the recognized objects and associated properties are displayed in the scenario data pane. The Gantt chart has now picked up and displayed the activities associated with the mechanized brigade’s Fix operation. Note that the aggregate Fix operation has been expanded to include implied activities. In this case, it is necessary for the brigade to first move to the enemy regiment’s location, and then engage in the actual Fix operation.

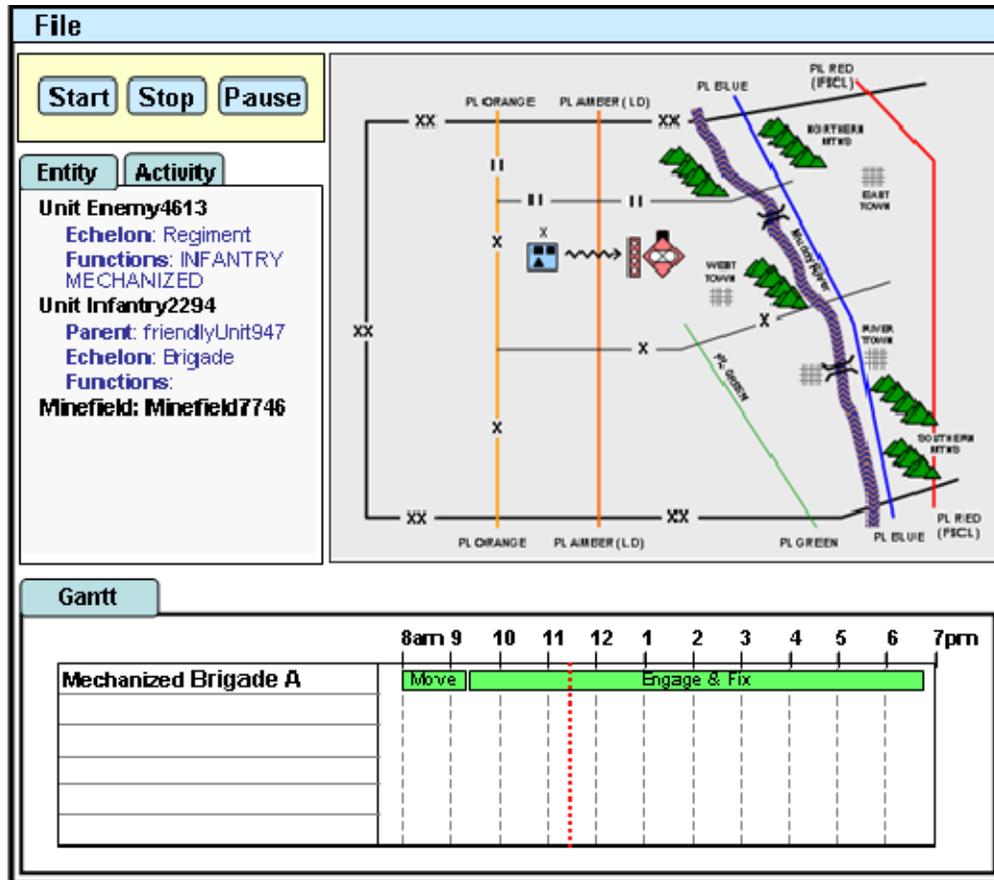


FIGURE 3-3 – Aggregate Operation

Complete COA Sketched

The complete COA is drawn in this panel, with the addition of three friendly units, one enemy unit, three operations, and an objective. The three new friendly units have been picked up in the Gantt chart, along with their planned activities.

At this point, the user is free to continue editing either the sketch, or the Gantt chart, until the desired operational intent has been captured. When satisfied, the user may simulate the plan by depressing the Start button.

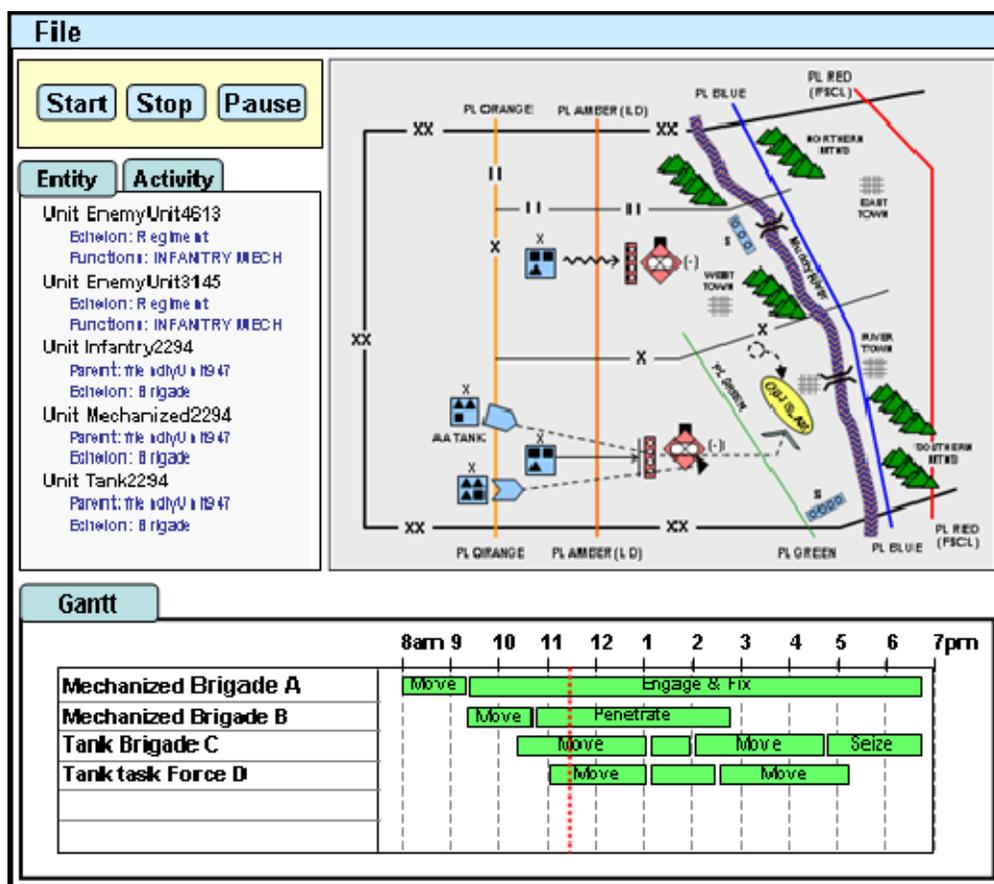


Figure 3-4: Complete COA Sketched

4 Software Architecture

The three principal components of CCOAT, along with the simulation, and interfaces, are shown in the Figure 4-1. COA design proceeds as the user's pen strokes are picked up by the GUI and sent to Magic Paper, which recognizes them as MIL STD 2525b symbols, which are then sent on to the Core for further processing. Other user inputs, such as Gantt chart timing adjustments, are sent directly to the Core. Experiment control directives are sent to the simulation. The Core returns COA information that has been sketched back to the GUI, for display in the Gantt chart, or scenario data pane. Once the user is satisfied with the drawn COA, the Start button will signal the Core to send the developed COA on to the simulator for execution.

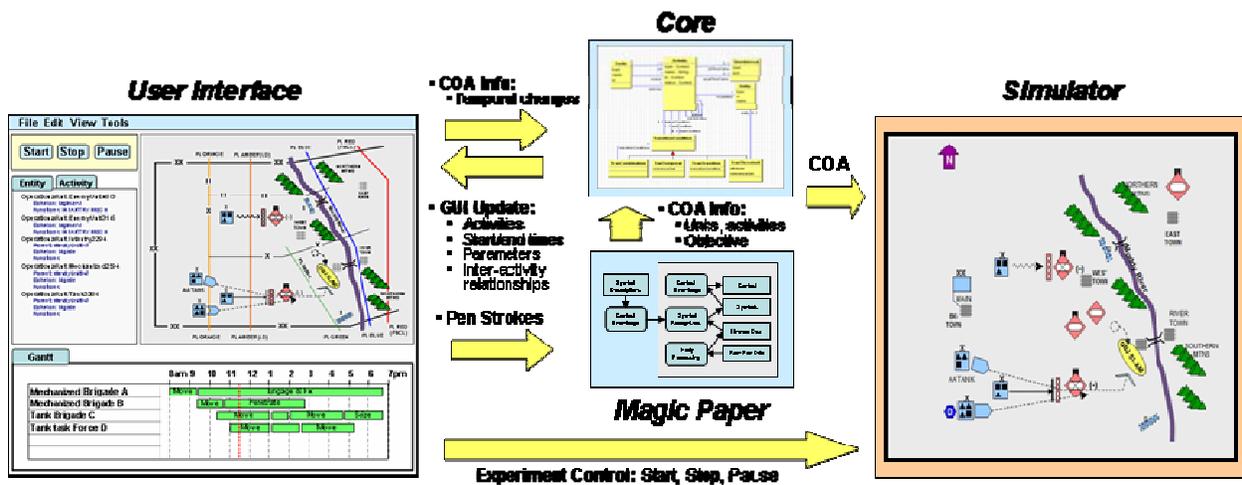


Figure 4-1: CCOAT Software Architecture

The various methods for communicating between CCOAT components, as well as the subcomponent's implementation languages are indicated in Figure 4-2. Communication techniques include XML over sockets, JNI, and an event notification technique for coordinating the GUI with the Core. In the later case, both GUI and Core listen for events from one another. When information in either domain is updated, a notification of this event is sent and the change is reflected in the other domain.

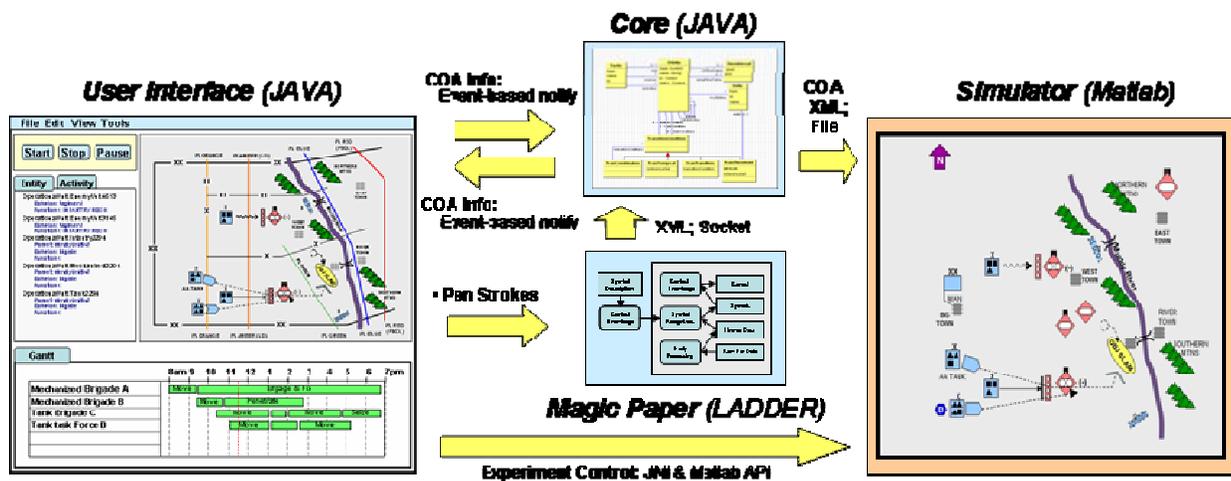


Figure 4-2: Implementation Languages and Communication Techniques

4.1 Sketch and Speech Recognition

There are three main components of the Magic Paper architecture: the sketch recognizer, the COA Domain Handler and the COA Multimodal Recognizer. The sketch recognizer is capable of recognizing shapes in the COA domain. Rules for constructing COA symbols are applied through the COA Domain Handler, which also provides an interface to other systems. The COA Multimodal Recognizer combines both pen and speech input and allows the system to receive details through voice input that might not be sketched easily. We describe the first two here; the multimodal recognizer is described in the next section.

4.1.1 The Sketch Recognizer

The sketch recognizer in turn consists of three components. As pen-input data is captured by the computer, the data is passed to the *primitive recognizer*. The primitive recognizer analyzes and classifies individual pen-strokes. Once an initial interpretation 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 Recognizer are combined in the Domain Recognizer in order to recognize complex shapes in the domain. Figure 4-3 shows the system architecture for the sketch recognizer.

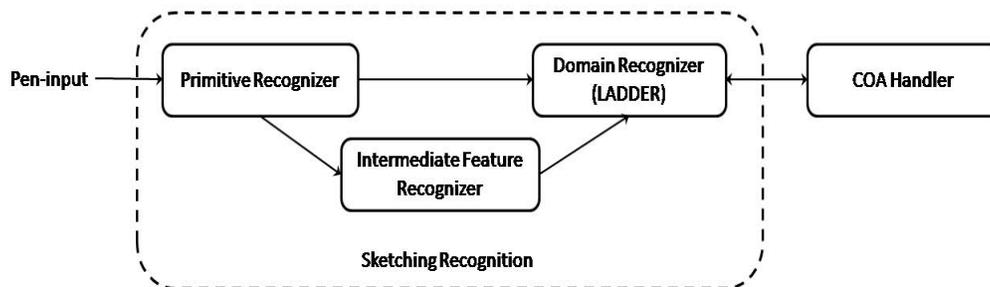


Figure 4-3: Sketch Recognizer Architecture

Primitive Recognizer

The tablet hardware provides data about the location of the stylus on the screen and its pressure, sampled 80 times per second, with each sample time-stamped. The hardware also packages up data into strokes, defined as the data collection between the time the pen touches the screen and when it is removed. Each pen stroke consists of one or more data points.

The primitive recognizers determine whether the pen stroke can be classified as an ellipse, line, point, polyline, or scribble (an erasure gesture). 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 4-4 shows a sequence of points collected by the pen that has been classified as a line, while Figure 4-5 shows a stroke classified as an ellipse. 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.

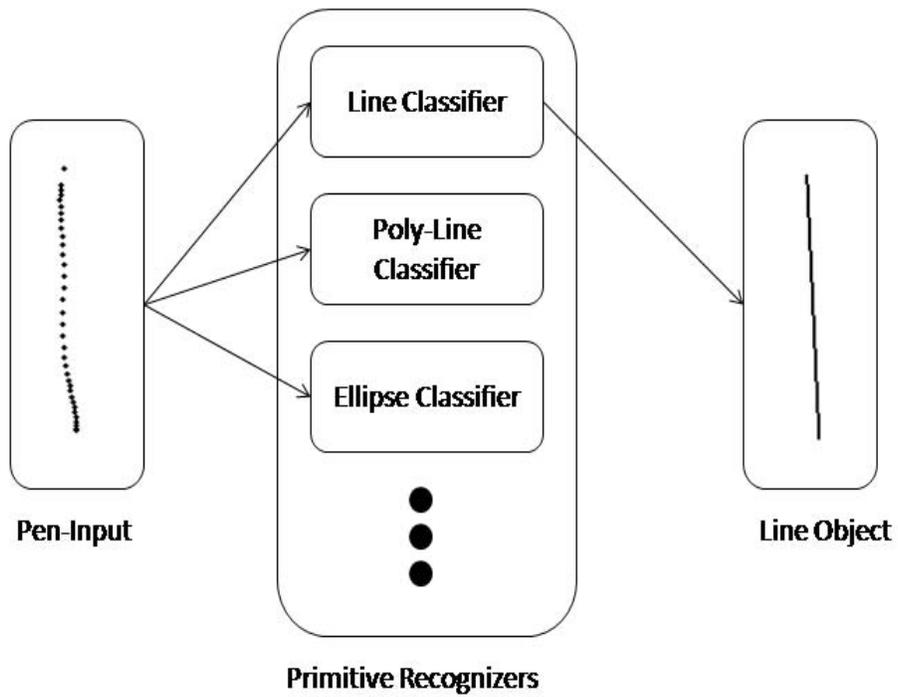


Figure 4-4: A raw stroke classified as a line.

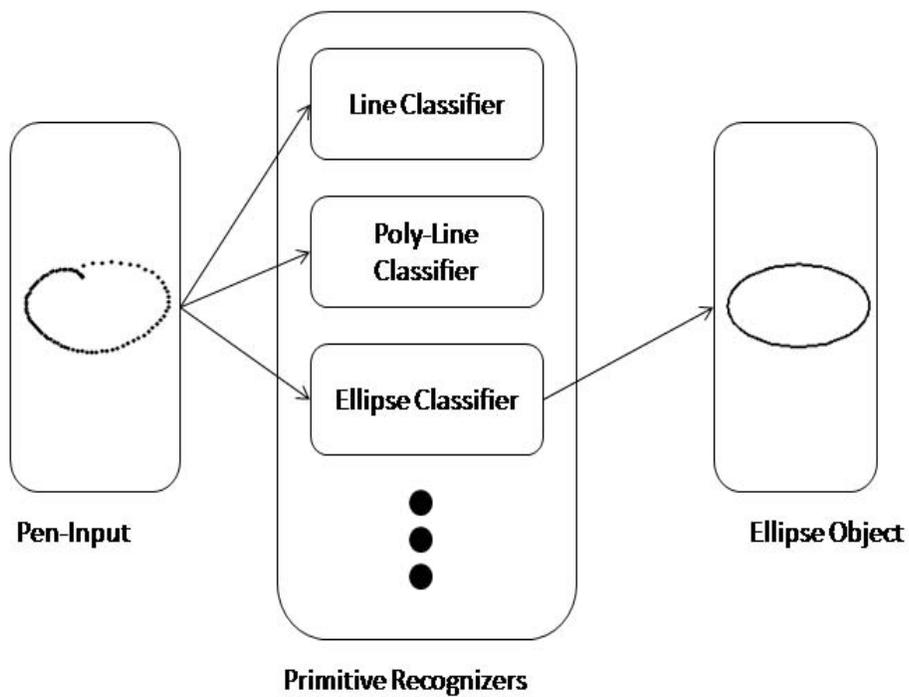


Figure 4-5: A raw stroke classified as an ellipse.

Because some classifiers have overlapping thresholds for acceptance, it is possible for a single stroke to be classified as multiple different primitives, provided the 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 contains details about which pen stroke it classifies, and the domain recognizer ensures that each pen stroke is (eventually) used only once in the final interpretation.

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, for example, 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; 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 simply the stroke. As noted earlier, multiple primitive classifications of a stroke are allowed, so a single stroke may result in several primitive objects getting 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, curve, arc, point, 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. The resulting line segments of the polyline are created as line primitives and sent to the domain shape recognizer.

Intermediate Feature Recognizer

The primary purpose of the Intermediate Feature Recognizer is to recognize shapes of intermediate complexity, i.e., those that can't be detected by either the primitive recognizer or the domain recognizer. The primitive recognizers handle shapes drawn with a single pen-stroke, while the domain shape recognizer deals with shapes defined by a fixed number of components. The intermediate feature recognizer fills the gap between these two, handling shapes drawn with a variable number of strokes, as for example dashed lines (planned movements), dashed frames (representing pending or suspected locations of units) and dashed ellipses (used to represent a seize action.) It does this by examining primitive objects returned by the Primitive Recognizer, looking for collections of these that form shapes of intermediate complexity.

Domain Recognizer

Magic Paper relies on the LADDER shape definition language [Hammond07], which defines shapes in terms of a list of components and a set of geometric constraints on those components. The components of a shape in turn may be either primitive shapes or other shapes in the domain.

The task of the domain recognizer is to recognize where any of the shapes in the current collection of LADDER definitions have been sketched. The collection of shapes recognized 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 be combined with other shapes in the VSC to produce a more complex shape. A shape is recognized 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.

Both the LADDER shape definition and our recognition system are 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. Recognition is incremental - the entire COA diagram does not need to be sketched - only those 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.

The search space for domain shape recognition can get quite large, because there can be many domain shapes, and each component of a domain shape may be paired with any of the recognized shapes that were added to the VSC. For each combination of domain shape description, component shape, and recognized shape, the geometric constraints for the shape must be evaluated to see if the recognized component shapes can be combined to form the domain shape. Since this is a large search space, we use the constraints as filters to prune the search space, removing a shape combination whenever it violates one of the constraints in a shape definition. For example, a shape may require two objects, each of type line, which have equal length. Recognition can be made reasonably fast by incrementally computing and storing lists of each shape type (e.g., a list of lines) and lists of objects that satisfy constraints (e.g., pairs of lines that are the same length).

4.1.2 COA Domain Handler

The primary job of the COA Domain Handler is to enforce rules for constructing symbols in the domain, rules that arise from the implicit grammar that governs COA symbols. As a simple example, the infantry symbol (diagonal lines) can be combined with the mechanized symbol (oval) to indicate mechanized infantry, and the artillery symbol (a dot) can be combined with mechanized, but infantry and artillery do not combine.

We use what we call frame templates to serve as a grammar to enforce these rules. Each type of frame template can combine with only specified types of unit modifiers. 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 ensures that the recognition of shapes follows the COA symbol language.

When a frame has been recognized, information about it is sent to the COA Domain Handler, which creates a template for each type of modifier possible for that frame. It then adds the templates to the vocabulary of shapes to be recognized. This ensures that a modifier will be recognized only in the context of a symbol to which it can validly be added. In addition, once a modifier is recognized as a valid modification to an existing frame, the frame templates for alternative frame templates are removed from the sketch recognizer, thereby ensuring that invalid combinations are never assembled.

Figure 4-6 shows a part of the hierarchy of unit frame templates. Vertical movement through the hierarchy indicates a valid addition to a symbol, while alternative branches without arcs show exclusive choices. Hence a basic rectangle can have either an infantry symbol added to it (the diagonals) or subunits (but not both). If the infantry symbol is recognized, the other frame

template branch is eliminated, but the user is still free to add, say, a mechanized symbol (the oval). As the arc indicates at the root of the tree, a unit symbol may have both modifiers and an echelon indicator. The hierarchy in Figure 4-6 indicates the rules; our implementation of the frame template mechanism allows modifiers to be sketched in any order.

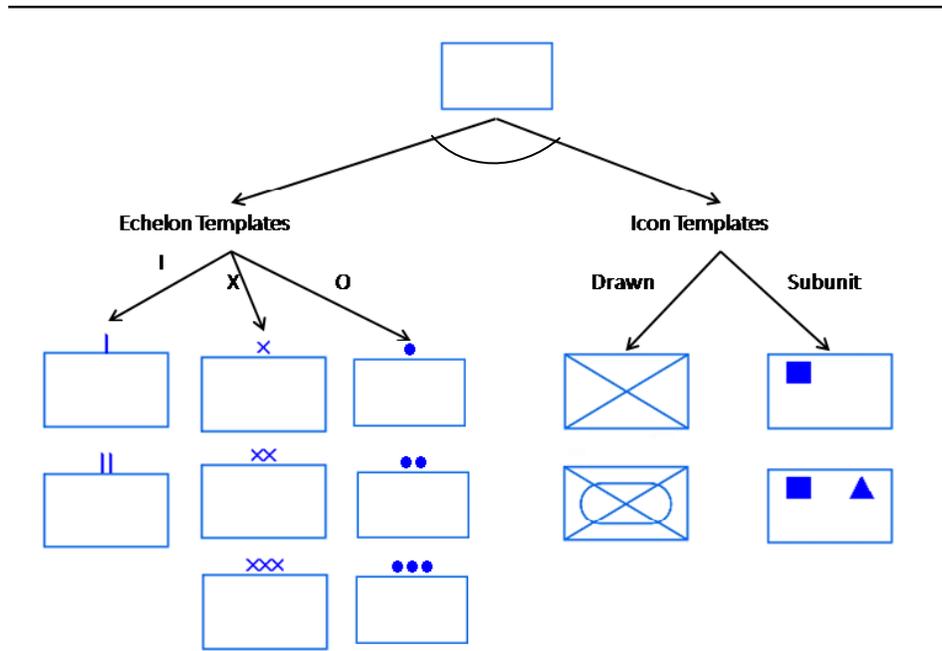


Figure 4-6: Part of the frame hierarchy.

4.1.3 Multimodal Recognizer

The multimodal recognizer is used to combine pen and speech input. We would ultimately like to build a system in which commanders explain their intent through an unfettered conversation with their team using speech and sketching as appropriate. In this model, the computer plays the role of a “fly on the wall” listening to the commander’s story and making sense of it in much the same way as a member of the command staff would. In this project, we have taken only the beginning steps to support this. Speech and sketching are captured through separate input channels and made sense of largely through separate interpretation processes; integration is achieved late in the pipeline, largely by resolving pronominal references in terms of sketched objects. For example, consider a case where the commander says: “this unit moves to that location” while drawing (or pointing at) an icon and then circling a location; then the multimodal recognizer should interpret the first pronoun as referring to the unit designated by the icon and the second pronoun as referring to the appropriate location.

The speech processing part of the system is provided by a relatively new version of the SUMMIT speech recognition system. SUMMIT has been under development in the MIT Spoken Language Systems group for nearly two decades and forms a key component of the Galaxy speech understanding system. The SUMMIT system uses many of the techniques used in all modern speech recognition systems: Hidden Markov Models, n-gram statistics, etc. It has

a particular emphasis on accurate segmentation of the acoustic signal before matching against a library of phoneme waveforms and the rest of the recognition process. It has been trained on an extremely large corpus of spoken language from both native and non-native and male and female speakers. It has high accuracy on continuously spoken language captured on low quality microphones (e.g. telephones) and does not require speaker specific pre-training.

The new version of SUMMIT is structured as a client-server system. The client initiates a session by providing the server with a grammar in the JSGF format. The client then repeatedly captures and digitizes the waveforms of each user utterance (at 8 khz) and transmits the digitized waveforms to the server. The server does all of the remaining acoustic and linguistic processing, producing a rank ordered list of the n-best interpretations of the utterance. (The client is light-weight (a few hundred lines of code) and is written in JAVA (there are other clients written in Python and other languages) while the server has many thousands of lines of C code).

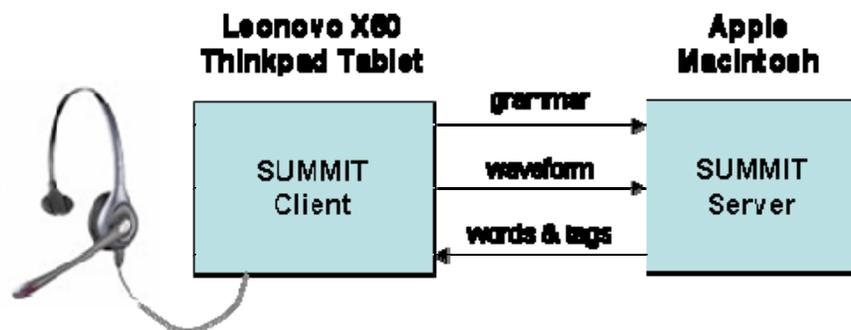


Figure 4-7: SUMMIT Client-Server Architecture

Humans often rely on the context of audio fragments and their understanding of the syntactic rules of a language in order to clarify speech. The SUMMIT system does this as well, using the grammar provided to SUMMIT at session initiation time as a domain specific language model that is combined with its background statistical models of English. A simple grammar for the COA Domain was developed as part of this project and is shown below.

Given the grammar and a digitized waveform, SUMMIT produces an n-best list of candidate sentences, each of which is accompanied by a score. This, together with the output of the sketch recognizer system is the input to the Multimodal Recognizer. The Multimodal Recognizer combines these two sources of information to determine the user's intent. The data from the speech recognizer consists of an n-best list of candidate sentences. The output from the sketch recognizer consists of one or more sketched shapes. These outputs are compared by the COA Multimodal Recognizer to determine which actions to perform.

As shown in the grammar below, 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 the given command of the n-best list, then the command is processed based on the input stroke.

```

<direction> = (north {tag-north} |
               east  {tag-east}  |
               south {tag-south} |
               west  {tag-west}  |
               )

<action> =    (moves {tag-move} |
               fixes {tag-fixes} |
               destroys {tag-destroys}
               )

<unit> = (regiment {tag-regiment} |
          battalion {tag-battalion} |
          brigade  {tag-brigade}
          )

<name> = (objective slam {tag-objslm})

<set-command1> = put a <unit> here {tag-put}

<name-command> = this is <name> {tag-commandName}

<copy-command> = copy this unit here {tag-commandCopy}

<motion> = this <unit> moves to the <direction>
           and <action> the enemy {tag-move}

```

Figure 4-8: Sample COA Grammar

4.1.4 Tactical Actions

Some actions consist of an aggressor and a defender. For these actions, the sketched shape often consists of an arrow-like shape, examples of which are the Penetrate and Fix action (see Figure 4-9). We use the geometric properties of directional properties of the arrow (start of the shaft, end of the shaft, direction the arrow is pointing) to determine the intended aggressor and defender. A common LADDER shape definition of “tail” and “head” is used in all shape descriptions for actions of the type aggressor/defender, allowing the COA Domain Handler to locate the endpoints of the arrow and determine the direction the arrow is pointing.

The COA Domain also contains actions that indicate a unit tasked with following another tactical operation, such as the Follow-and-Assume and Follow-and-Support actions. In Figure 5, for example, the tank-heavy brigade in the south is tasked with following and assuming the attack launched by the mechanized brigade, while the tank task force is charged with following and supporting the tank-heavy brigade. These actions contain a base symbol (indicating the type of action and beginning location of the action), and a path that indicates related actions. The LADDER shape description for these shapes similarly indicates a base shape and a dashed or solid line indicating a path. The line indicating the path of the action may contain multiple segments or waypoints. In order to determine the related action for the “Follow and...” actions, the shortest distance from the path to all other actions is analyzed to determine the action that is to be followed. The unit doing the following is determined by the closest unit to the base of the “Follow and ...” Action.

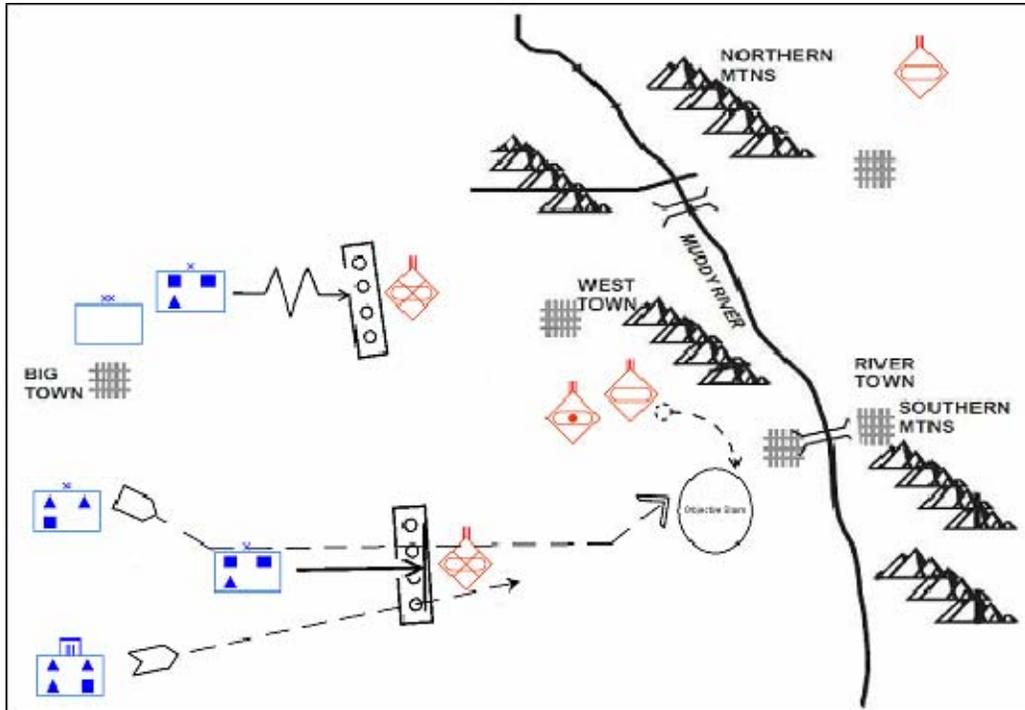


Figure 4-9: A COA sketch with a follow-and-assume and a follow-and-support action in the south.

4.2 Core

The CCOAT Core is responsible for maintaining the data associated with a sketched Course of Action, and for providing that data to the Graphical User Interface and the Simulation Engine as needed. The Core receives data from Magic Paper that describes objects and activities that have been sketched. Using internal data representation models, the Core is able to augment the information provided by Magic Paper with data supplied by, or derived from, the models to produce much more detailed object descriptions. For example, this includes attributes of units, such as min and max speed over various terrain, as well as expansion of aggregate activities, such as Fix and Penetrate, to more detailed activities.

4.2.1 Data Models

The data representation models provide a foundation for the CCOAT Core that allow it to process, manage, and generate the individual components of a Course of Action description. The data models for this project were developed in UML, the Unified Modeling Language. The models describe the objects that comprise a Course of Action, and define the specific properties of each individual object as well as the relationships between objects. There are two major types of objects described by the models. Entity objects represent discrete tangible things such as tanks, aggregate tangible things such as minefields, and intangible things such as a location or an area. Activity objects represent an action or a set of actions taken by one or more Entities; examples of Activity types include Move and FollowAndSupport.

Activities

The Tactic data model is centered on the “Activity” object (see Figure 4-8). An Activity includes properties such as a name, a unique identifier, and a status (e.g. Planned, Completed, etc.). Each Activity also specifies the Entities that participate in the Activity, and the role that each Entity plays. Two Time Intervals may also be associated with an Activity. The first, called the “Estimated Time Frame,” is generated by the CCOAT Core at the time the Activity is created or modified. This represents the Core’s best estimate of when the Activity will nominally begin, and when it will nominally finish. Activity durations may be specified directly in the models, or they may be derived from other aspects of the Activity (e.g. the duration of a *Move* activity is calculated from the distance traveled and the speed of the object that is moving). The second Time Interval is called the “Actual Time Frame,” and can be filled in by the Simulation Engine after completing a simulation run of the Tactic. Finally, each Activity may include a description of the conditions that cause the Activity to begin, or to terminate. These conditions, called *TransitionConditions*, may be temporal (e.g. start at a particular time), value-relative (e.g. end when location equals x), or a combination of multiple conditions.

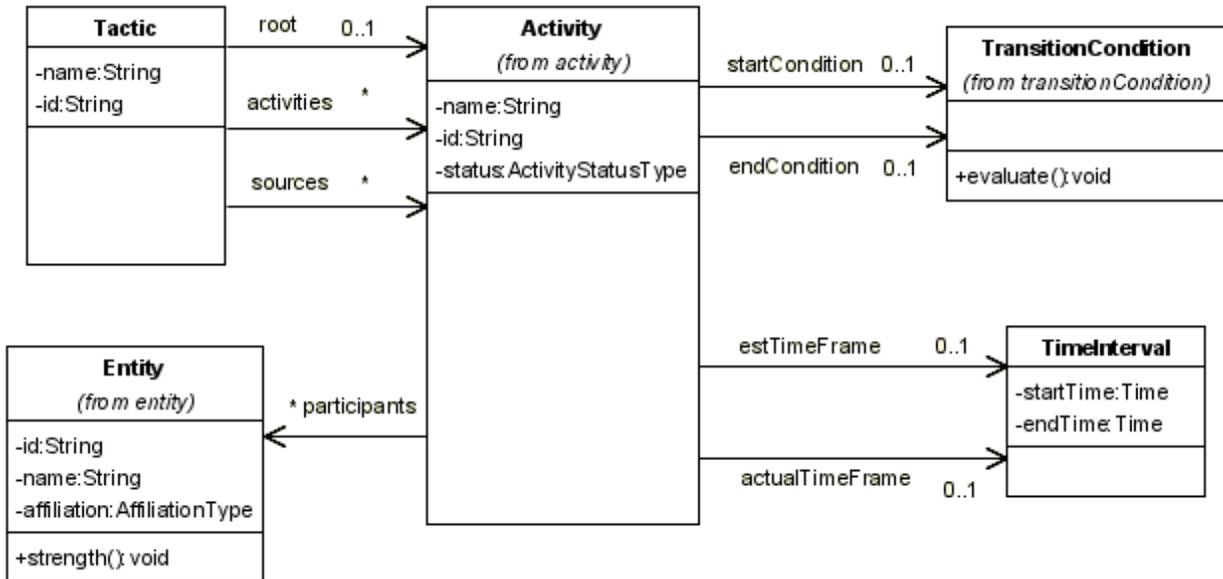


Figure 4-8: Activity Model Details

An Activity may represent a single, discrete action to be taken, or it may represent an aggregation of multiple actions. One function of the CCOAT Core is to decompose any aggregate Activities received from Magic Paper into a sequence of atomic Activities that can be executed by the Simulation Engine. For example, when the aggregate activity Penetrate is sketched in Magic Paper, the CCOAT Core will generate the atomic activity sequence {Move, Penetrate} to indicate to the Simulation Engine that one Entity must change its location before initiating the Penetrate action. For this Seedling, the specific Activity decomposition rules were directly encoded in the software. However, they could also be specified as UML process diagrams related to each aggregate Activity class, or as a set of structured directives that are interpreted by the CCOAT Core.

Entities

The second major class of modeled objects is Entities (see Figure 4-9). Similar to Activities, an Entity may represent either a single, discrete object, or an aggregation of multiple objects. The

Entity model follows a “composite” pattern, where certain types of Entities are specifically noted as being composed of other Entities. The generalized composable Entity, called *Composite*, has a list of associated sub-components that are the lower-level Entities. A *Composite* Entity may be made up of other Composite Entities, thereby allowing a hierarchy with as many levels as necessary. As with Activities, each Entity specifies both a name and a unique identifier. Entities also specify their affiliation. More specific Entities may also define their own properties; a *MineField*, for instance, must specify the type of *MineField*, and whether or not the mines are scatterable.

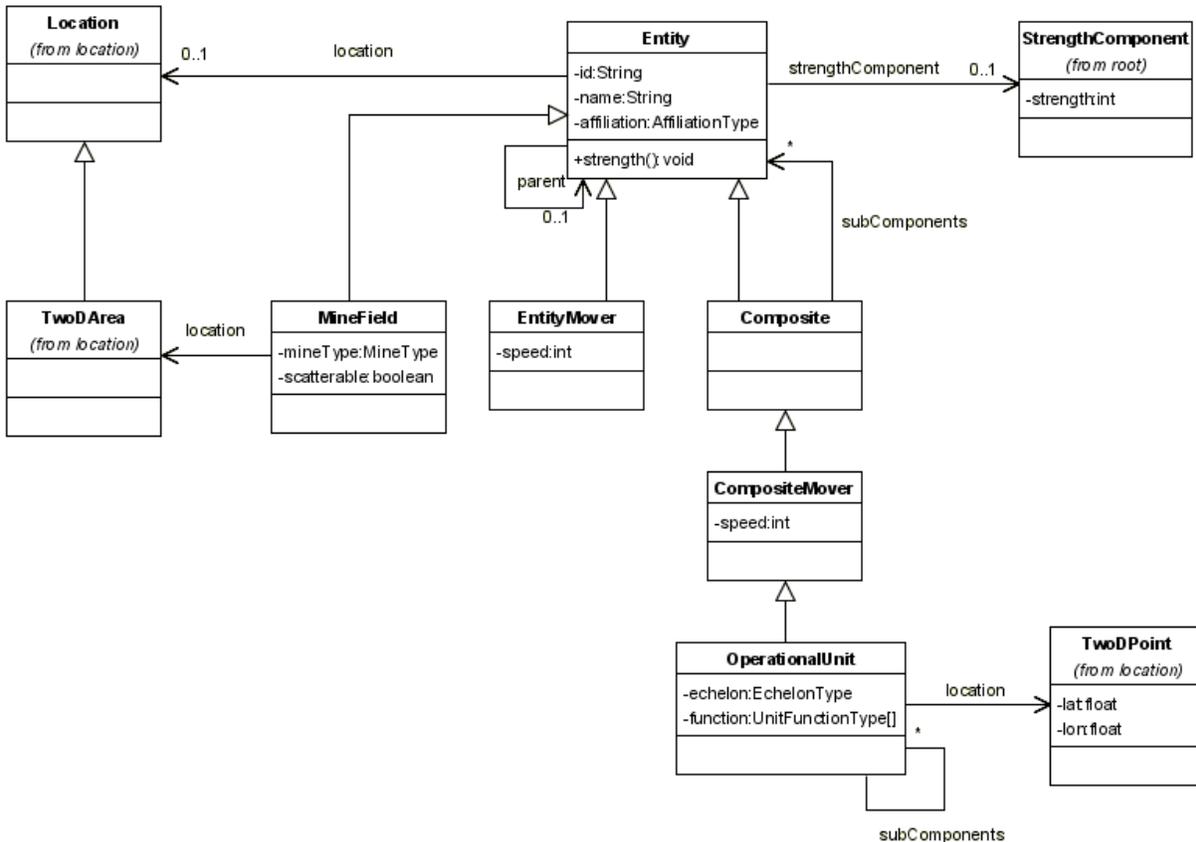


Figure 4-9: Entity Model Hierarchy and Details

4.2.2 Architecture

The CCOAT Core is organized into four main functional blocks, shown in Figure 4-10. One block receives and processes data generated by Magic Paper. Another handles the decomposition and sequencing of activities according to the data models. A third functional block serves as the primary data container for the Tactic objects, managing and maintaining the Entities and Activities, as well as controlling access to the data from the other functional blocks and from the User Interface. The fourth functional block is responsible for generating XML data describing the Tactic that can be processed by the Simulation Engine.

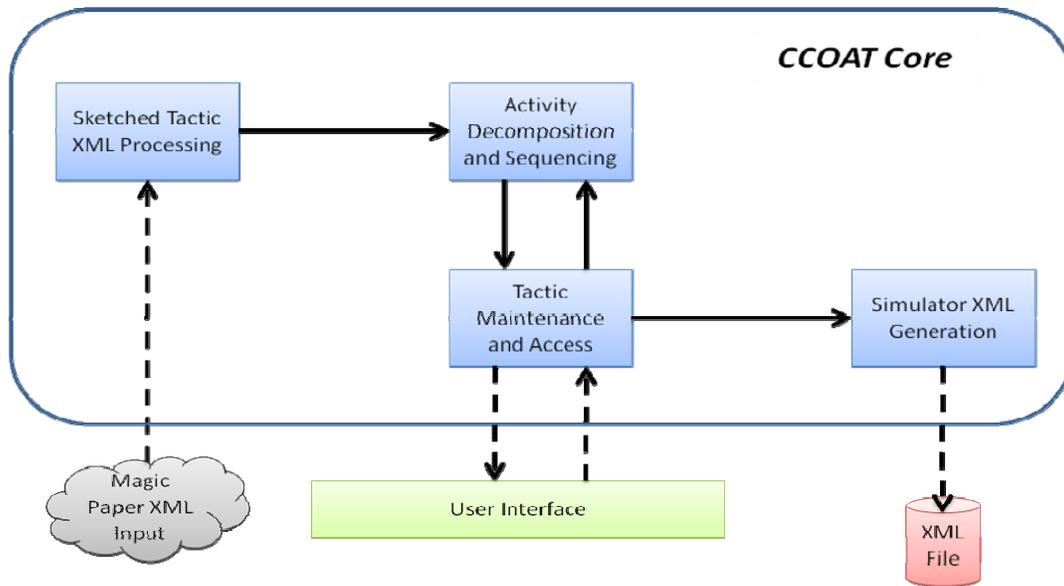


Figure 4-10: CCOAT Core Functional Architecture

The CCOAT Core communicates with Magic Paper through an asynchronous TCP socket interface. The sketched data is represented in XML, the eXtensible Markup Language. The XML representation generated by Magic Paper closely matches the UML data model for Activities and Entities; even though the sketching interface cannot generate all of the information required by the data model, the XML interface with Magic Paper is flexible enough to be able to carry only those pieces of information that Magic Paper is capable of recognizing in the sketch and generating in the XML. See Figure 4-11 and Figure 4-12 for examples of the XML specification of particular objects in the data model.

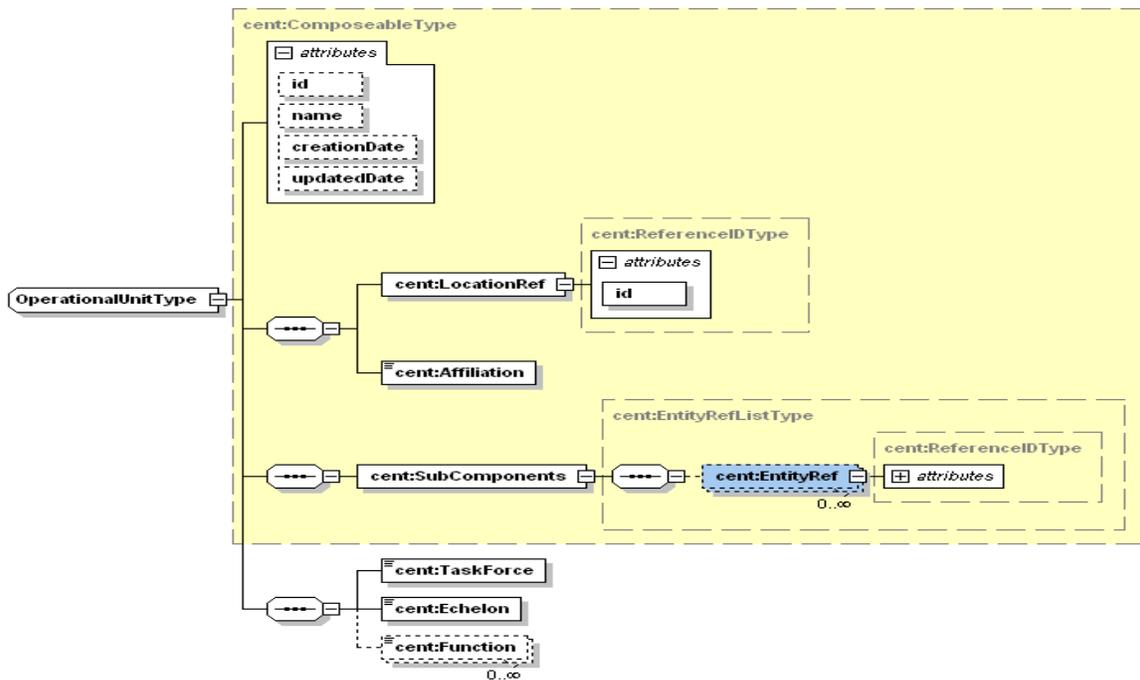


Figure 4-11: Entity OperationalUnit XML Definition

The same XML structure is also used to carry the data across the interface between the CCOAT Core and the Simulation Engine. The data provided to the Simulation Engine is fully compliant with the data model. When signaled from the User Interface, the Core writes all of the data for the current Tactic to a text file, and the User Interface starts the Simulation Engine by telling it the location of the data file.

After receiving new data from Magic Paper, the CCOAT Core processes the input data and updates the internal representation of the sketched Entity or Activity, creating or modifying objects as necessary. Newly-sketched Activities are sent to the Activity Decomposer, which is responsible for generating the set of atomic Activities that can be executed by the Simulation Engine. The atomic Activities are then sequenced and added to the Activity chain for a particular participating Entity. As part of the sequencing, the Core determines the conditions that indicate when each Activity must begin, and the conditions that must be satisfied for the Activity to terminate successfully. These conditions may be temporal (e.g. start at time x , terminate after two hours), value-relative (e.g. terminate when location is near y), or a combination of the two. Furthermore, for each atomic Activity, the Core estimates the time that the Activity should begin and terminate, based on its start- and end-conditions and the estimated times of other related Activities.

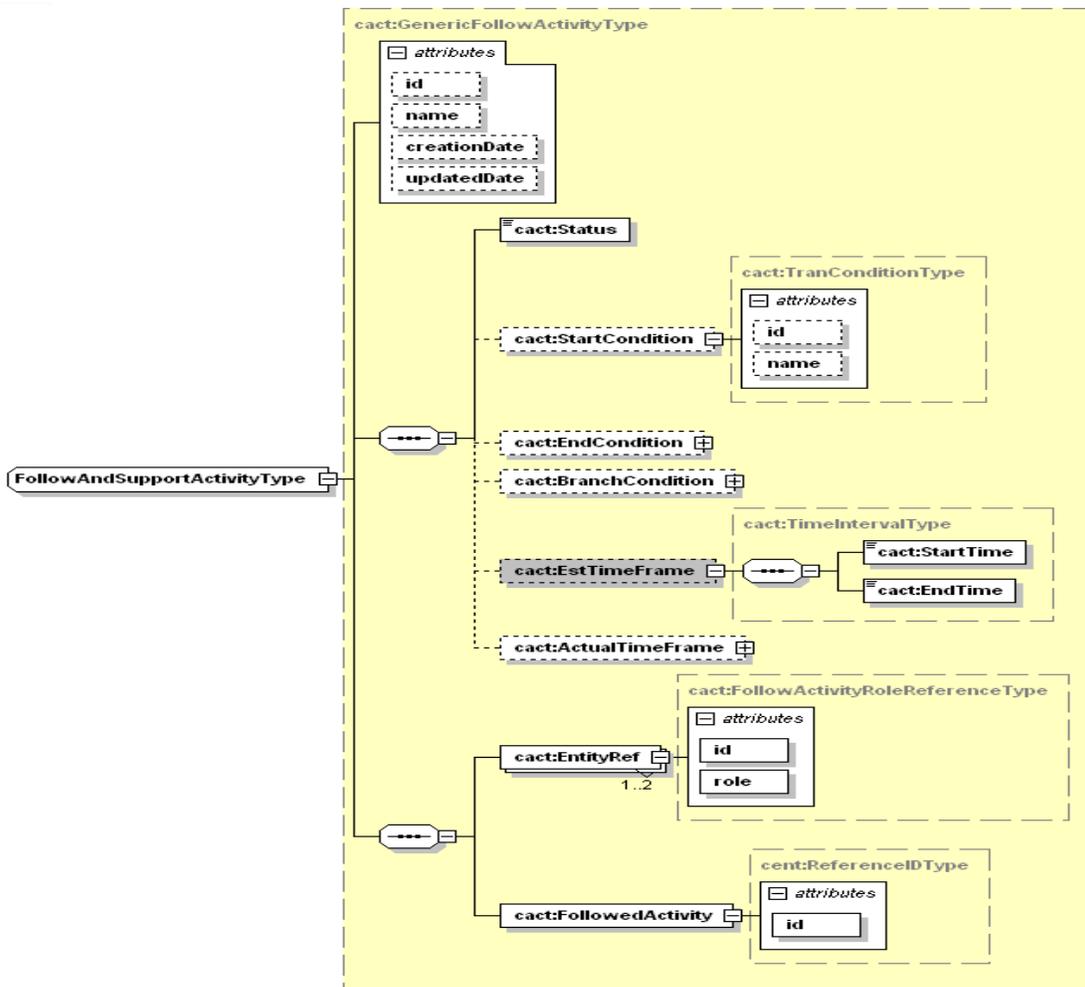


Figure 4-12: Activity FollowAndSupport XML Definition

4.3 Graphical User Interface

A snapshot of CCOAT's actual user interface is shown in Figure 4-13. Here, the user has sketched a few blue and red units and associated operations. The rough sketch strokes have already been cleaned up by the sketch recognition software, yielding a crisp presentation. The sketch panel and underlying Magic Paper software have already been discussed in Section 4.1.1 and will not be discussed further here. The Experiment Control panel, as well as the Scenario Data panel, currently have limited functionality and will also not be elaborated here. This section will focus mainly on the Gantt chart: the information it displays, and the information it obtains from the user.

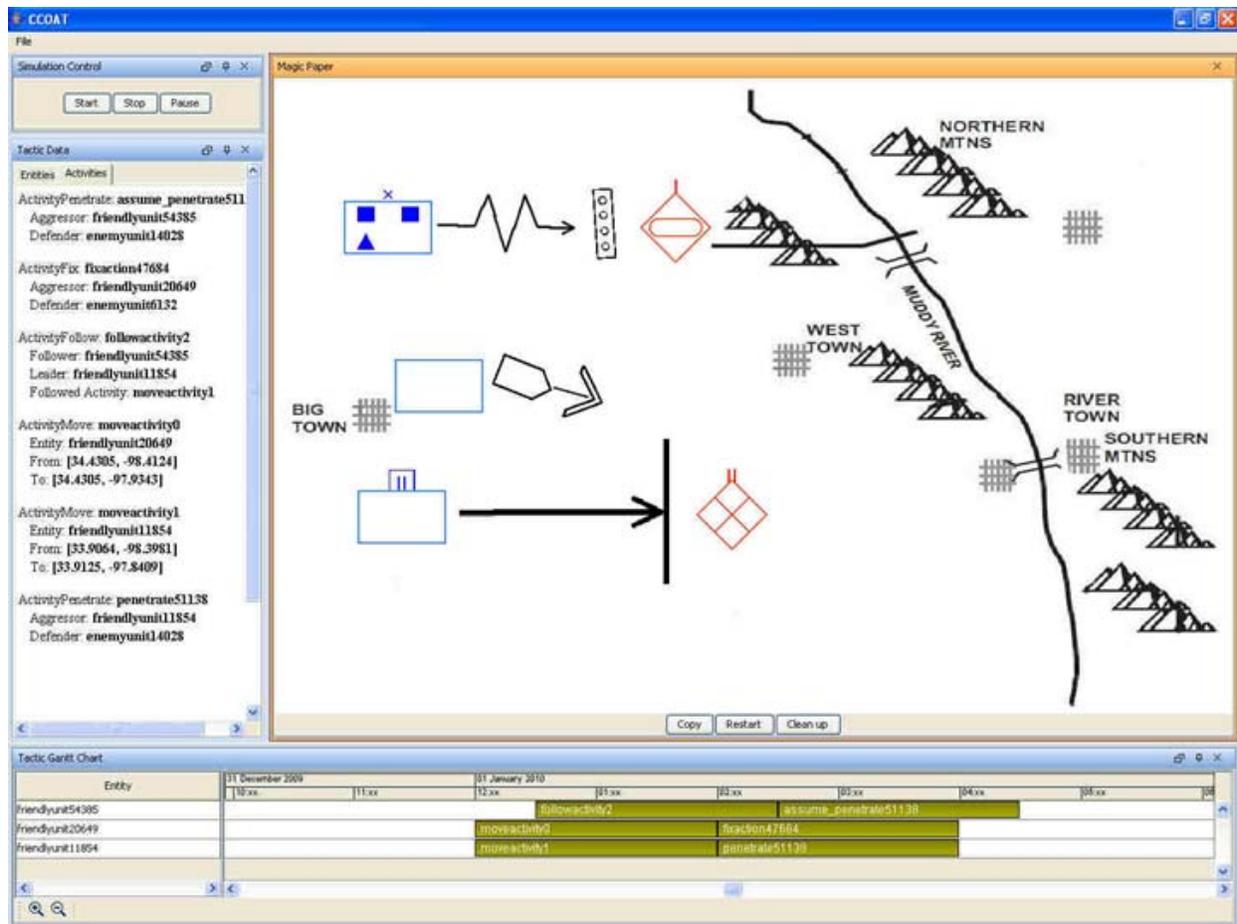


Figure 4-13: CCOAT GUI

4.3.1 Gantt Chart Panel

A Gantt chart is an intuitive and familiar way to comprehend and interact with the temporal aspects of activities and activity sequences, and is the primary method for displaying and modifying this type of information in CCOAT. The Gantt chart in Figure 4-13 is magnified slightly in Figure 4-14.

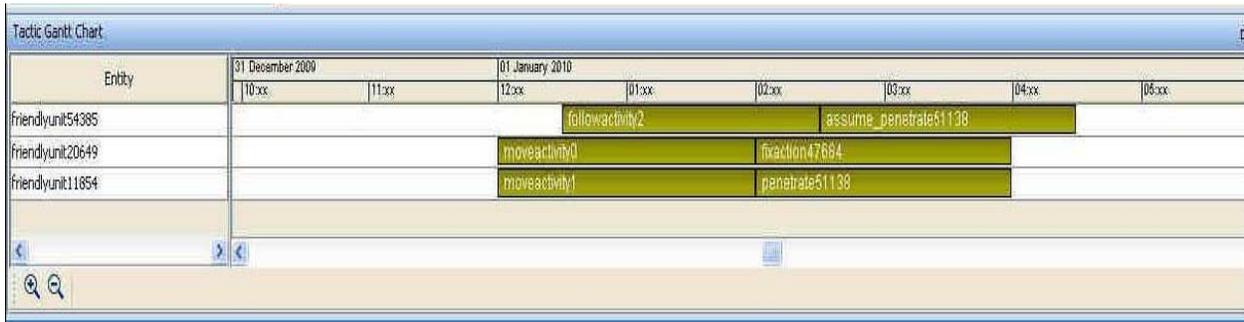


Figure 4-14: CCOAT Gantt Chart

Each row in the Gantt chart displays the activity sequence of a particular unit; three units in this case. Of course, the Gantt chart merely reflects the information that the CCOAT Core has developed for the COA. Though not completely implemented at this time, the Core estimates the duration of each activity, using various physical models and also activity models. The duration of Move activities, for example, must be in accordance with the distance a unit moves, and the speed over the intervening terrain. In another example, Fires type activities may have a fixed duration that is specified in an activity model.

The CCOAT Core also estimates the start time of all activities. The relative start times between and among activities are often well defined by transition conditions that are associated with the process models that the Core references in building up the COA. For example, it may generally be true that activity 'A' immediately follows activity 'B' in time. The Gantt chart would then show these activities abutting one another on the time line.

The actual start time for a sequence of activities may be difficult, or impossible, for the Core to estimate, because there is not enough information in the sketch to correctly fix the sequence on the time line. In this case, the Core starts the sequence at the earliest possible time, which is generally the current time, now. The user is then free to adjust the timing of an activity, or sequence of activities, by sliding them left or right, using the stylus.

COA Information Not Displayed

Gantt charts are good at displaying the activities in a COA, their position on the time line, and in certain cases, the relationship between two activities. The relationship where an activity is to start upon the completion of some other is often quite apparent, as we see in the sequences in Figure 4-14. More complex relationships among activities that are defined through Boolean relationships on the state of the world, or with lead/lag relationships between/among activities, are not so easily displayed. They do exist, however, and are an important part of the COA. Various symbols, colored in different ways, may be a way of at least alerting the user to the existence of relationship. A pop-up dialog could then be provided that describes the relationship, and possibly a way of editing it.

Another aspect of a COA that is not currently displayed, though doing so would be straightforward, is that of branch plans. Plans that fork off into two or more branches would be displayed in the Gantt chart on separate lines in the chart.

Editing COA Activity Timings

Using a pen stylus, the user can drag activities in the Gantt chart earlier or later in time, as well as modify their start and/or end times, assuming the underlying inter-activity temporal constraints allow such manipulation. Temporal changes made in the Gantt chart are, of course, reflected in the CCOAT Core.

Cursors appearing, as the stylus hovers over activities in the Gantt chart, provide visual feedback regarding temporal adjustments that can be made. Two examples are shown in Figure 4-15, where (A) an entire activity, or activity sequence, is dragged left or right, or (B) the start or end time of an activity is modified.



Figure 4-15 A: Dragging an Activity



Figure 4-15 B: Adjusting an Activity's Start Time

Constraints on Editing

Typically, activity start and end conditions depend on a wide range of world events, as well as on start and completion times of one or more other activities. So a web of constraints and dependencies across the activities can hamper the arbitrary manipulation of their timings. Dragging one activity may push or pull one or more other activities along with it, due to these dependencies, or there may be limits on the extent to which timings can be modified. Indeed, it may not be possible to modify some timing at all, as they may be fixed.

There are two classes of constraints that will impact editing: those that define when activities can begin and end (transition conditions), and those that are model based. As we've said, the first class describes a web of inter-dependencies that must be honored, as activity timings are altered. The second class focuses on the duration of activities that may be constrained by either physical models, or activity models. For example, changing the duration of a Move activity changes the speed at which the participating unit must move. A unit's speed is naturally constrained by its inherent maximum speed, the terrain over which it must move, the weather at the time of the

move, and perhaps other factors. Activity models try to capture sensible, or doctrinal, constraints that are difficult to discern from the physics of the situation. The Commander may, for example, wish to conduct a fires operation, over a fixed period of an hour, no more, no less. In this case, the Gantt chart cannot allow the user to modify the duration.

4.3.2 Other Panels

Text and Data

The data panel displays textual information about the sketched entities and activities. Currently, the user is able to observe the software slowly build up an understanding of each symbol, as parts of it are sketched and recognized. Much additional information about entities and activities, their properties and state, could be displayed, as well as allow modifications by the user. Further enhancement of CCOAT would allow such capability.

Simulation Control

This panel in the prototype provides basic control for starting, stopping, and pausing the simulation.

Magic Paper

The Magic Paper sketcher is implemented as a Java panel, allowing straightforward integration with the main GUI window.

5 Hardware Architecture

The CCOAT prototype hardware configuration is shown in Figure 5-1, where a separate computer is employed to host the SUMMIT speech server. All other CCOAT software is run on the Thinkpad tablet.

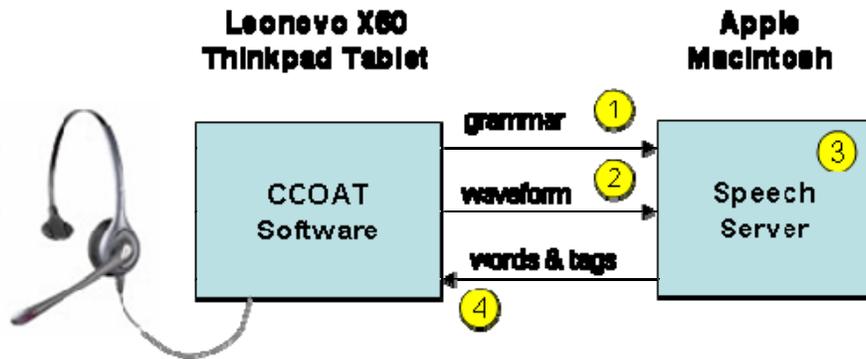


Figure 5-1: CCOAT Prototype Hardware Architecture

The coordinating steps between the two computers are as follows:

1. Grammar (in JSGF format) passed to SUMMIT Speech server during session initialization. Semantic tags are included in grammar.
2. Client (in Java) digitizes waveform and passes it via xml-rpc to server
3. SUMMIT server processes waveform against the session grammar
4. Server returns n-best list of recognized words and tags to client

Views of the Leonovo X60 Thinkpad tablet, showing its two basic configurations, are shown in Figure 5-2. Its screen can be flipped up, rotated, and then folded down for convenient use as a tablet. Specifications for the Thinkpad used in the Seedling project are:

- Tablet XP
- 12.1 WVA SXGA+ TFT display
- 100 GB 7200RPM disk
- 4GB RAM
- Intel Core Duo L7400
- UltraBase, DVD recordable 4X, UltraBay ethernet
- Integrated 1Gb ethernet
- 11a/b/g/n Wi-Fi wireless LAN Mini-PCIE
- 8 cell Li-ion battery



Figure 5-2: Leonovo X60 Thinkpad Tablet Computer

6 Simulating Courses of Action

The simulation helps the user verify that the sketched Course of Action has been understood by CCOAT. It also enables a level of evaluation to be performed, with respect to outcome, duration of the operation, resource usage, etc. It was not the purpose of this research to develop a highly accurate simulation, or even to integrate such a simulation with the CCOAT application. The simulation developed for this study required only a small amount of effort, by design, but served to a) construct an end-to-end CCOAT system and b) show that the sketch was properly interpreted.

The simulation's design centers on entity and activity data structures, similar to CCOAT Core's internal organization. The activities drive the simulation, dictating the behavior of participating entities. The main simulation loop continually monitors the running activity list, initiating and terminating activities as appropriate and evolves the entity state over time, according to well-defined, but flexible models.

The following subsections discuss the simulation's logic flow, types of data maintained, activities modeled, and the animation display.

6.1 Execution Flow

The simulation's execution flow is shown in Figure 6-1, where initialization is followed by an execution loop that terminates when either formal end conditions are satisfied (described below) or until a pre-specified time has been reached.

When CCOAT application launches:

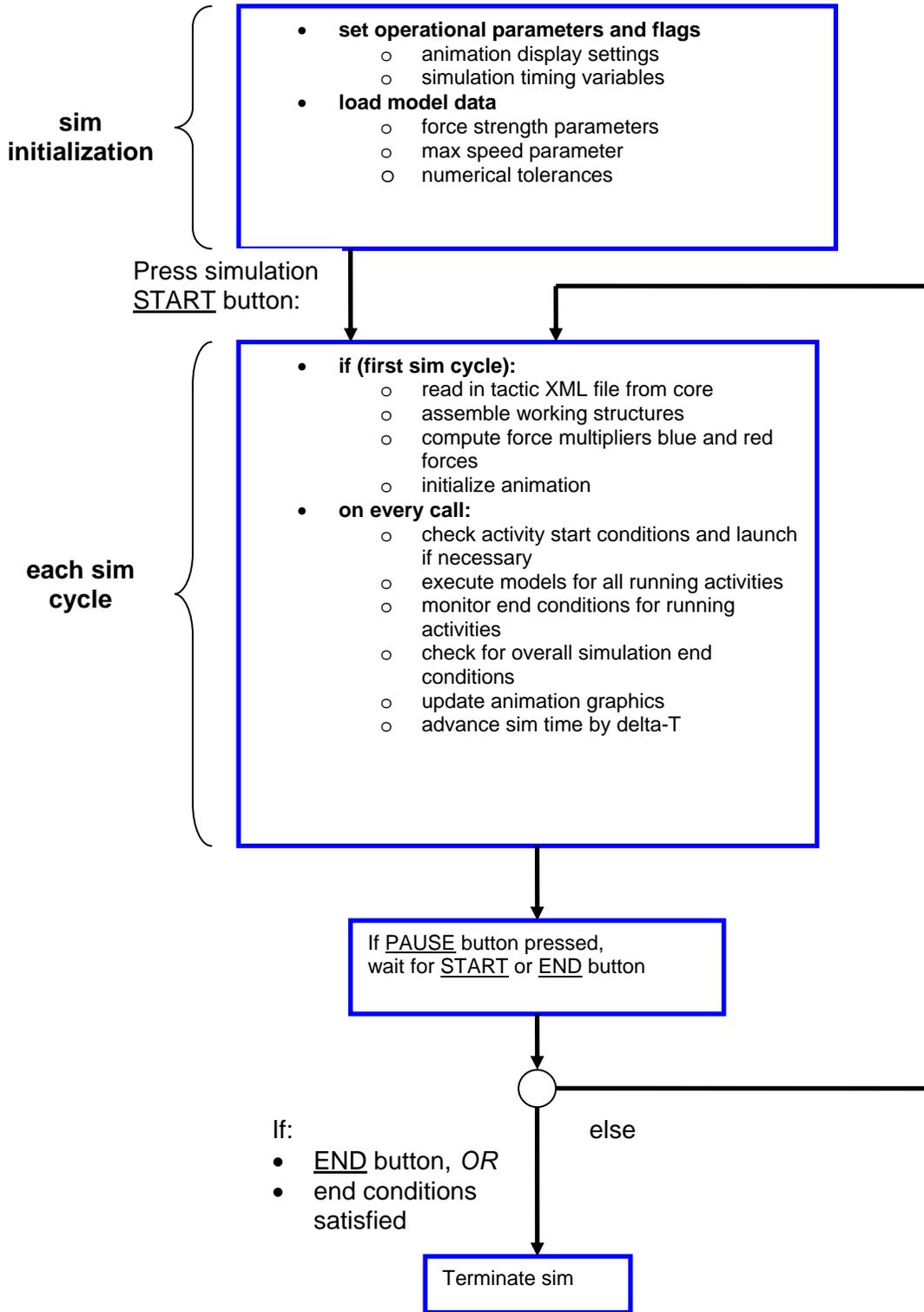


Figure 6-1 - Simulation Execution Flow

Initialization

Initialization occurs when the CCOAT application is launched. At this time, animation parameters and tactics model data are initialized for later use, such as simulation cycle delta-T interval, information display flags, and model data that indicates how force multipliers (FM) are computed.

First Cycle

Upon the Start button being pressed, CCOAT Core outputs an XML file to the simulation. This file describes all the entities and activities which comprise the COA, as well as necessary location data for each force entity and minefield. The simulation then begins executing the individual cycles over small delta-T intervals. On the first cycle, a few one-time operations are executed. These include: reading in the XML file, parsing it into Matlab-readable data structures (using a shareware tool available from MathWorks), converting this data into the working entity and activity structures that drive the simulation, computing force multipliers for each unit, and launching the animation.

Execution Loop

The following operations are executed on each cycle:

1. **Check PLANNED Activities** - Check the start conditions for any PLANNED activities (i.e. not-started); set the activity status to RUNNING if the conditions are satisfied.
2. **Check RUNNING Activities** - For each RUNNING activity, evolve the state of each participating entity, over the delta-T interval, using modeling information.
 - **Check End Condition** - For each RUNNING activity, check to see if the end condition is satisfied. If so, set the activity status to COMPLETED.
3. **Check Simulation Termination** - Check the simulation termination conditions. Normally this is when all planned activities have been either COMPLETED or have otherwise been removed from consideration, due branches not taken, for example.

6.2 Data Structures

The simulation's data structures and their attributes are listed below. Dynamic attributes are shown in italics.

Blue and Red Force Units:

- identification tag
- affiliation (friendly or enemy)
- echelon
- subcomponents (lower echelons)
- functionality (mechanized, infantry, armored, etc.)
- *current activity*
- *current location*, given as a (lat, lon) pair
- *current strength, or FM* (an entity is "dead" if its FM is zero)

Minefields:

- identification tag
- affiliation

- type
- scatterable Boolean
- detonation period (how often a mine may detonate)
- likelihood (i.e. probability) of a detonation at the detonation time
- statistics for how much damage occurs from a detonation (given as a standard deviation in FM units)
- *active/deactivated status*
- *number of current mines*
- coordinates of boundary vertices

Activities:

- identification tag
- type
- *current status*: e.g. "PLANNED", "RUNNING", "COMPLETED"
- start condition type
 - full description of the start condition parameters
- end condition type
 - full description of the end condition parameters
- participant(s)
- any additional information specific to that activity type:
 - e.g. for a move: the start and end locations
 - e.g. for a Fix or Penetrate: who the aggressor and defender are
 - e.g. for a follow: who the leader and follower are
- branch condition parameters, if the branch exists

6.3 Activity Types and Behaviors

Activities that are supported by the simulation are listed below, along with a brief description of the logic that is executed when the activity is in the RUNNING state.

MOVE activity:

1. Identify the mover and current (lat, lon) coordinates.
2. Identify the goal (lat, lon) of the move.
3. If the move is just beginning, record the activity start time.
4. Check if the moving entity is in a minefield.
5. if this is the first entity entering the minefield, note this fact.
6. Compute the speed of the entity; this is either the default maximum speed for unobstructed motion, or a reduced value of that maximum if in a minefield.
7. Advance entity towards the goal location by changing the current (lat,lon).
8. Monitor if it is time to consider a mine denotation, stochastically determine if the mine blows up and how much damage occurs, and attrit the occupying entity accordingly.
9. Reduce the number of undetonated mines by one for each explosion.
10. Declare the minefield cleared, i.e. deactivated, once the first entity that entered leaves; this behavior is such that entities entering the minefield region later on will not be slowed or attrited.

- Evaluate end condition: declare the move COMPLETED if the entity is within some numerical radius of the goal (lat, lon) location.

FIX activity:

- Identify the Fix aggressor or defender.
- Verify that combatants are within a reasonable fighting radius.
- If the Fix is just beginning, record the activity start time.
- Every “fix period” evaluate the combat results table as follows:
 - Compute the force ratio of aggressor-to-defender based on their current FMs.
 - Stochastically roll a “twelve-sided” die, uniformly distributed over the number of rows (sides) in the table, 1-12. (Rolling two six-sided die would not yield the desired uniform probability density.)
 - Look up the outcome of the current combat cycle (go to the table row corresponding to the die outcome; go the column corresponding to the force ratio, interpolating between columns, if necessary).
 - A *positive* table value corresponds to a percentage FM attrition of the *defender*.
 - A *negative* table value corresponds to a percentage FM attrition of the *attacker*.
 - Attrit the damaged force accordingly.
 - Evaluate end condition: end the battle after a fixed battle duration.

	5:1	4:1	3:1	2:1	1:1	2:3	1:2	1:3	1:4	1:5
1	10%	8%	-1%	-1%	-2%	-2%	-5%	-10%	-20%	-30%
2	11.8%	9.1%	0%	-0.5%	-1.6%	-1.7%	-4.5%	-9.3%	-18.8%	-28%
...
12	30%	20%	10%	5%	2%	1%	1%	-2%	-6%	-8%

Table 6-1: Example Combat Results Table for Fix Action (aggressor-to-defender FM ratios across the top; die roll values along the rows)

Penetrate activity:

- Penetrate activity behaves identically as a Fix activity, but with a combat results table more skewed towards aggressor success.

Follow activity:

- The Follow activity behaves identically as a Move activity, but has its own specialized start condition (discussed below).

Support activity:

- Identify the supporting entity and what activity is being supported (in the current simulation, the only supported activity type is a Penetrate).
- If the support is just beginning, record the activity start time.
- Transfer the supporting entity’s FM to the supported attacking entity.

4. When the supported Penetrate is complete, re-divide the combined FM evenly between the supporter and supportee.
5. Evaluate end condition: support is complete if the supported Penetrate is complete.

Before listing the various start condition types for activities, it is important to mention two subtleties of the simulation. First, for the Support activity, a transfer of FM occurs between two force entities as they combine their attack on a hostile force. In this case, there is only one evaluation of the combat results table every battle period. It is also possible that multiple forces could fix or penetrate a hostile force without explicitly combining their resources. This situation typically occurs during a Follow-and-Assume operation. In this case, the combat results table is evaluated for each individual Fix or Penetrate activity at every combat period.

The second point involves the so-called branch condition, which can be appended to any activity. A branch condition evaluation occurs when the activity end condition is reached, and takes an outcome value of either “true” or “false”. This branch will determine what the next activity will be, depending on some current status of the overall simulation state. For example, after the follow portion of a Follow-and-Assume tactic, it may be necessary to determine if a given hostile force entity is either strong or weak. If the hostile force is strong, then the next activity should be a Penetrate; if the hostile force is weak, a Move activity to some other location of interest may be appropriate. A branch condition is appended to the concerned subsequent activity’s working structure, so that their nominal start conditions are augmented with the need for either a “true” or “false” branch condition outcome. As such, branch conditions are useful for achieving “forks” in the overall COA activity flow.

The start conditions are in general not activity-specific (the follow activity is an exception), and will be listed afterward. The possible activity start conditions are:

- Temporal start conditions, e.g. the activity starts when a given simulation elapsed time is reached
- Completion of some other activity (e.g. a Fix begins when a move to the battle location is accomplished).
- An activity starts a designated period of elapsed time *after* the start of the "followed" activity.

6.4 Animation

Animation of the simulated COA is achieved through use of basic Matlab plotting and graphics commands. A snapshot of the COA, midway through its execution, is shown in Figure 6-2. Note that the units are showing their FM values, and the minefield locations are indicated with dashed outlines).

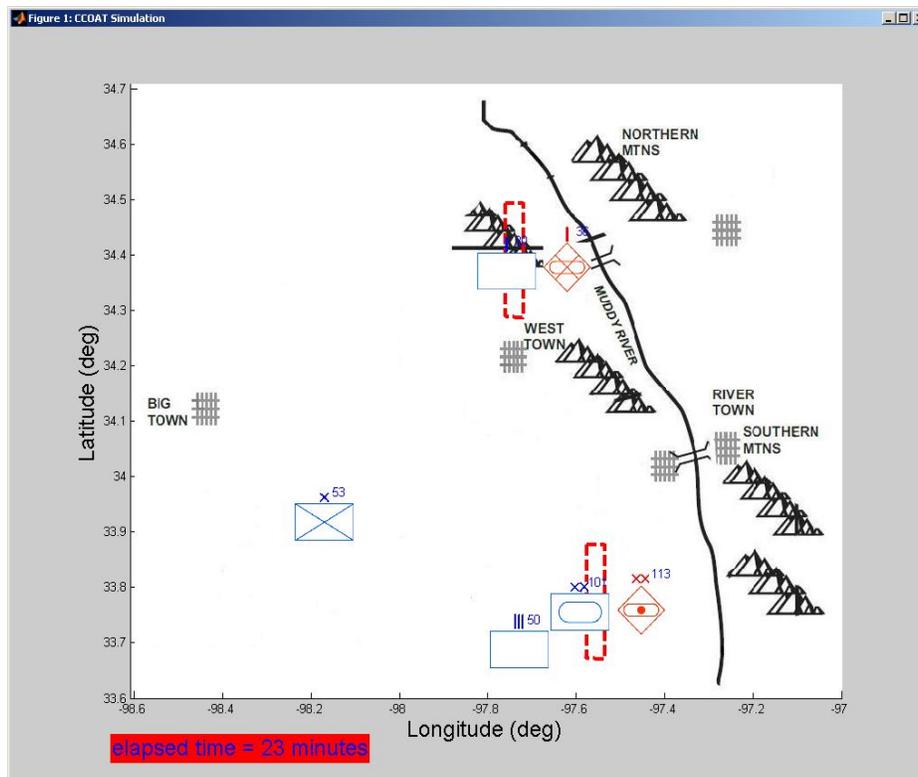


Figure 6-2: Animation Window

The simulation executes a number of functions upon its first cycle, and others there after:

First execution cycle:

- Set all the basic graphics parameters and settings (object scalings, resolution, image buffering, color, etc.).
- Load in available image files (entity function icons, echelon icons, map, etc.).
- Assign an appropriate icon/image to each entity.
- Loop over each entity, plotting their initial location and, if desired, their initial FM.
- Plot the appropriate echelon symbol (XX, X, |||, ||, etc.).
- Plot each minefield with an icon symbol and a polygon outline.
The polygon depends on the minefield vertices sketched in Magic Paper.
The outline color and thickness depends on the minefield status (active, inactive, or planned).
- Generate a list of plotting object handles for all the entities and their associated objects.

- Plot the initial clock value, if desired.

Subsequent execution cycles:

- Loop over each entity and update its plotted location and, if desired, the FM value.
- Change the minefield outline if its status has changed between active, inactive, and planned.
- Update the elapsed time clock value, if desired.
- Display a "simulation complete" message when the simulation end condition has been satisfied.

6.5 Monte Carlo Evaluation

Once a Coarse of Action is specified, it can be run repeatedly in the CCOAT simulation in a Monte Carlo batch-style test. Each time it is run to completion, certain stochastic variations will lead to slightly different outcomes. Over a sufficiently large Monte Carlo population, the commander can get an idea of the range and likelihood of possible outcomes for a given COA.

In the current CCOAT simulation, the primary stochastic variations occur from two sources: (1) random mine detonations when an entity is traversing/clearing a minefield, and (2) simulated die-rolling when evaluating combat results tables for the Fix and Penetrate activities. Both of these factors will directly affect the instantaneous force multiplier (FM) value of the participating entities. The accumulated effects of these force multiplier degradations can affect the overall battle outcomes for Fix and Penetrate activities and have branching effects on key COA decision points. For example, a Follow-and-Assume activity may branch depending on whether a certain enemy force, already being penetrated by another friendly entity, is either strong or weak when the following force approaches it. If the enemy is strong, the following force may linger in order to attrit the hostile force. However, if the enemy is weak, the following force may move on and "assume" some other task, especially if it is important for overall mission success. The Monte Carlo simulation will help the commander determine if the enemy force is likely to be strong or weak when the following force arrives and what the following force is most likely to actually do in practice.

Figures 6-3 through 6-6 show the outcome of 200 Monte Carlo runs of a single course of action, involving four friendly entities, two enemy entities, two minefields, and at least one Follow-and-Assume with a conditional branch. The variable outcome data includes final force FM, final force location (a latitude-longitude pair), and the number of mines remaining in a minefield. In this family of runs, enemyunit1234 is destroyed (indicated by the tiny final FM values) in 100% of runs. Its final location is always the same, since enemy forces do not move in this version of the CCOAT application. Alternatively, enemyunit9793 has a fairly wide range of final FM values, with a highest likelihood value of around 25. In some unlikely cases, enemyunit9793 will be almost completely destroyed, as indicated by the low outlier final values of around 10 and 4.

Consistent with its follow-and-assume conditional activity structure, friendlyunit36168 has a pair of possible final locations at the end of the COA. In approximately 60% of the runs (~120 of 200 cases), it ended up at (lat,lon) ≈ (33.8 deg, -97.59 deg) which corresponds to sticking around to help attack a strong enemy currently under penetration by another friendly force. In

the other 40% of cases, friendlyunit36168 moved north-northwest to (lat,lon) \approx (34.25 deg, -97.61 deg) to attack another hostile entity, since the penetrated enemy force was already weak. Note that since friendlyunit36168 was generally in supporting roles for attacks, its final FM remained fairly strong, around 50, in almost all cases. Finally, the histogram for minefield27339 shows the (integer-valued) number of mines remaining after it has been traversed and deactivated. Since this type of minefield is known to have 33 mines (based on simulation parameters), one can infer that traversing the minefield with no detonations is very unlikely. The most likely case is that 2 mines will detonate, giving the commander a feel for the expected mine risk associated with his COA.

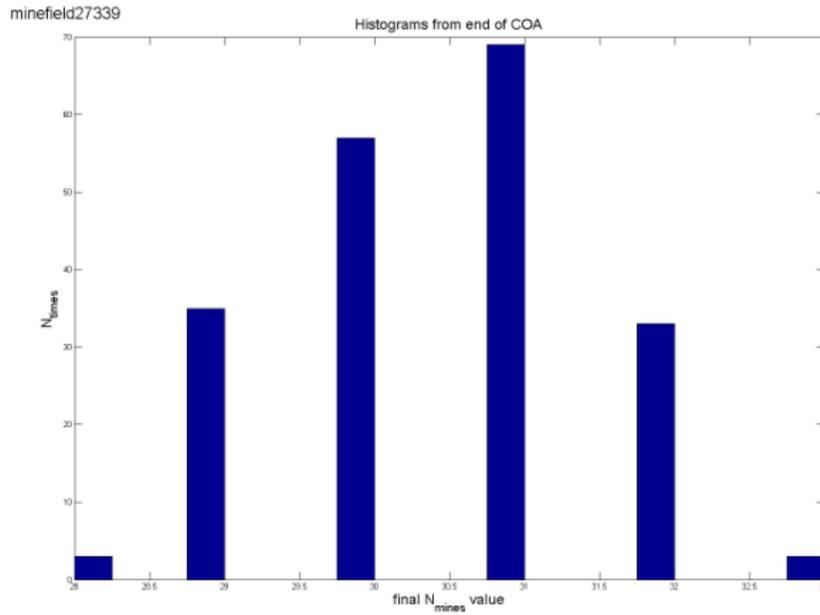


Figure 6-3

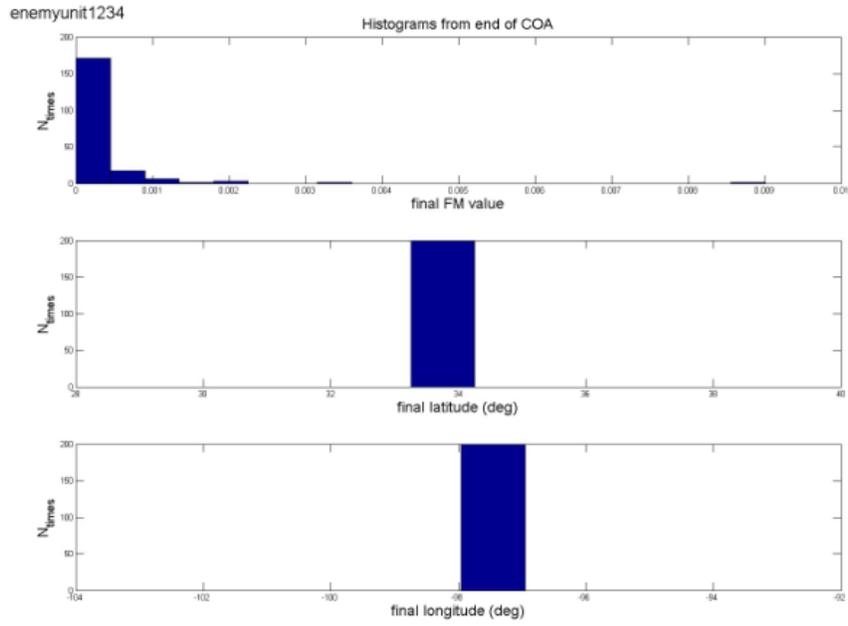


Figure 6-4

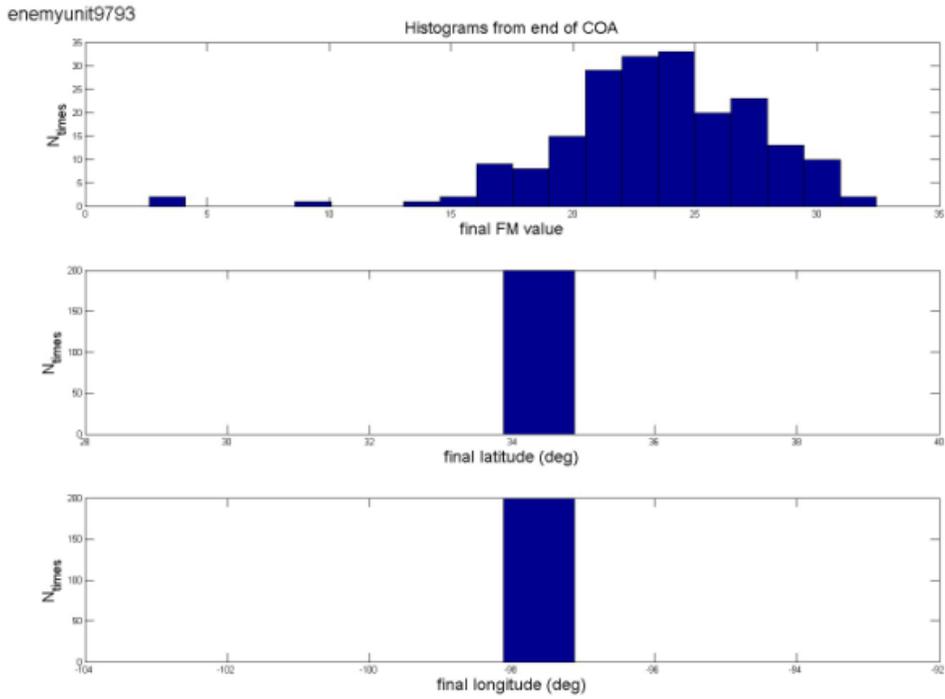


Figure 6-5

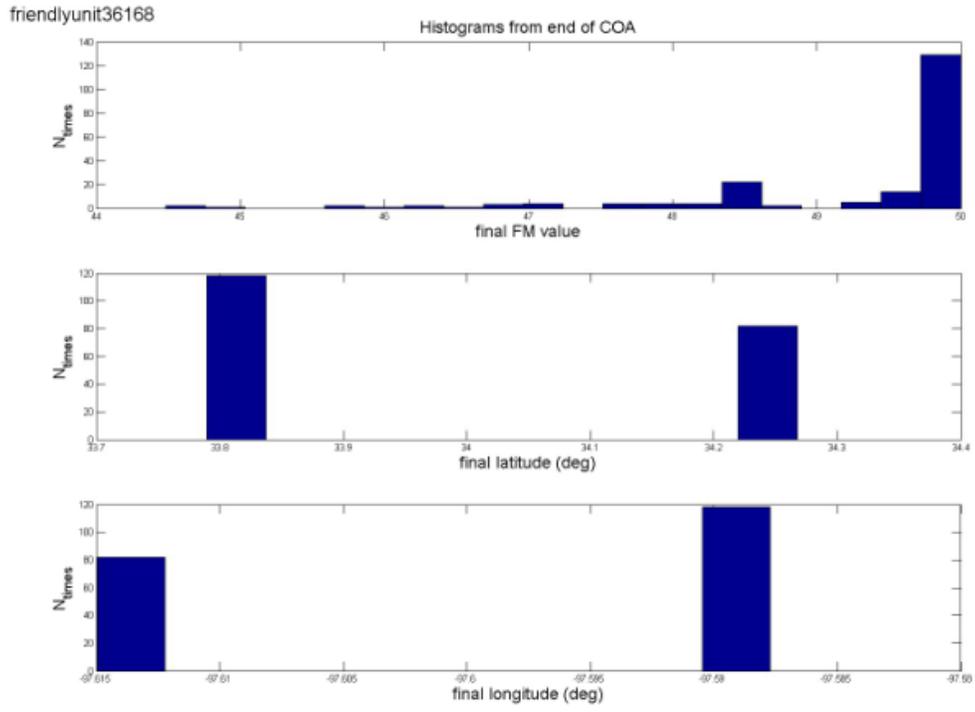


Figure 6-6

7 References

- Tracy Anne Hammond. LADDER: A Perceptually-based Language to Simplify Sketch Recognition User Interface Development. PhD Thesis for Massachusetts Institute of Technology. Cambridge, MA, January 2007.
- Zue, V., Glass, J., Phillips, M., and Seneff, S. 1989. The MIT SUMMIT Speech Recognition system: a progress report. In Proceedings of the Workshop on Speech and Natural Language (Philadelphia, Pennsylvania, February 21 - 23, 1989). Human Language Technology Conference. Association for Computational Linguistics, Morristown, NJ, 179-189. DOI=<http://dx.doi.org/10.3115/100964.100983>