a CApella: Prototyping Context-Aware Applications by Demonstration

Ian Li
CSE, University of Washington, Seattle, WA 98105
ianli@cs.washington.edu

Summer Undergraduate Program in Engineering Research at Berkeley (SUPERB) 2003

Faculty Mentor: Prof. Anind Dey
Graduate Mentor: Jason Hong

EECS, College of Engineering,
University of California Berkeley

August 7, 2003
Abstract

In recent years, various projects have developed infrastructures and toolkits to facilitate the development of context-aware applications. These efforts made creating context-aware applications easier for programmers, but not for creative designers and end users, most of which do not program. To help non-programmers, we present, a CAPpella, a tool that allows users to build context-aware applications by demonstration. There are two challenges to this approach: (1) the machine learning technique to deduce the users’ context and (2) the design of the graphical user interface to allow users to assist the training of the computer. This paper describes our design of a user interface using principles that aid in visualization of information. We discuss the results of our paper prototype test that shaped our design decisions and describe the final prototype and future work.

1 Introduction

The goal of a CAPpella is to support programmers and non-programmers to develop context-aware applications. “A context-aware application is [an application] that adapts its behavior to a changing environment” (Harper et. al., 2002). An example of a context-aware application is an application that turns on the lights when a person enters a room.

In the past few years, programming a context-aware application such as a lighting system has become easier due to efforts to create infrastructures and toolkits. Unfortunately, the development of context-aware applications is still limited to programmers and professionals. iCAP (interactive Context-aware Application Prototyper) allowed non-programmers to develop context-aware applications by using a visual language to create rules and to define functionality (Sohn and Dey, 2003). a CAPpella takes a different approach by allowing the user to demonstrate to the computer what the user wants the computer to perform. For example, Alice works in an office with a context-aware lighting system. She wants the lighting system to turn on the lights every time she walks into the conference room. To program the system, Alice starts the system to record her demonstration. Alice demonstrates by going into the room and turning on the lights. Alice performs several demonstrations, until the computer learns that every time she walks into the room, it should turn on the lights. Other examples of context-aware applications include:

- Security applications. Suppose Alice wants her alarm system to be turned on when she is sleeping or when she leaves the house.
- A shopping reminder application. This application can remind users to buy an item on their shopping list when they are near the item of interest.

Creating a CAPpella presents two challenges: (1) the machine-learning technique to create the classifiers that will deduce the users’ context and (2) the design of the graphical user interface to allow users to assist the training of the classifiers. This paper is about the latter. I will briefly describe the machine-learning technique that a CAPpella uses to explain the need for a user interface and I will describe the design of the user interface.
2 Background

Machine-learning algorithms that follow the classical machine learning (CML) model is impractical for supporting the creation of classifiers by non-programmers, because selecting the correct features to create a good classifier is difficult and training a classifier is prohibitively slow to be interactive. The classical model requires the user to pre-select the features that will be used to learn a classifier and it assumes that the training process is performed only once, thus the features need to be selected carefully. For many classification problems (e.g., identifying objects in an image), selecting the right features to produce a good classifier is an art in itself, requiring technical knowledge that non-programmers do not have.

To circumvent the problem with CML, a CAPpella uses the interactive machine-learning (IML) model (Fails and Olson, 2003). In the IML model, the initial step (feature selection) is integrated with the training of the classifier in an interactive loop. The IML model removes the initial step of CML by making the learning algorithm perform two functions: to learn the classifier and to filter the set of features that produces the best classification, thus relieving the user of the responsibility to know the best features for a classification problem. In the middle of this loop is the user who corrects the classification by creating more training data. In this model, the training time is designed to be fast, so the user can iteratively improve the classifier.

The design of the user interface complements the IML model’s fast interaction and feedback loop. The design follows graphic design principles in presenting data so users can quickly select the best features for classification.

The next section is a high-level description of an interaction with a CAPpella. Next is a discussion of the design principles behind our design decisions. Next, we present the
results of our informal paper prototype test. We then describe the user interface that applies our design decisions. We demonstrate how the interface complements IML with an example scenario. We conclude with descriptions of other features that can improve the user interface and future evaluation work.

3 Interaction Basics

This section elaborates on Alice’s task of programming a lighting system to turn on the lights every time a person enters a conference room.

An interaction with a CAPpella system starts with a session. A session is a demonstration of an activity in an area equipped with sensors. In Alice’s case, her demonstration happens in a conference room equipped with an overhead video camera, an audio microphone, a phone sensor, and a light switch sensor, all of which are hooked up to the system. Alice starts the system to record her demonstration. She demonstrates to the system by walking into the conference room, turning on the lights, and sitting on the table in the middle of the room. When Alice is done demonstrating, she stops the system from recording.

Events or states of an activity are sensed by the system and are recorded in logs. Each type of event has a corresponding log. In Alice’s demonstration, the events are: the number and location of people in the room; the sound volume; whether the phone is in use or not; and whether the light has been turned on or off. Although different sensors are used to sense a demonstration, events do not depend necessarily on the kind of sensor that is used. For example, in Alice’s case, video recording is used to get the number of people in the room, but RFID and other sensors can be used to get this information.

In a demonstration, the user also demonstrates actions. Actions are behaviors that the user wants the computer to perform and can be sensed and performed by the computer. In Alice’s case, turning on the lights is an action because the lighting system can turn on the lights. The machine-learning algorithm does not use actions to learn the classification of a demonstration. Instead, the actions are the behaviors performed by the system when it starts classifying activities. Suppose the machine-learning algorithm has learned the classification of Alice’s demonstration, the next time that she goes in the room, a CAPpella will take the actions (in this case, turn on the lights) and perform that action.

After a demonstration, the user uses a CAPpella to view the logs of events and to select the relevant events. To Alice, the relevant events in the logs are: when there is one person in the room; when the person is near the door; and when the light is turned on.

When the user has selected relevant events, the data are fed to the machine-learning algorithm for classification. The goal of the user is to iteratively select and deselect relevant events until the system is most certain of a classification. Suppose Alice has made other demonstrations with different classifications (e.g., a coffee break, a phone call, and a going home demonstration). The first time that Alice feeds the relevant events to the machine-learning algorithm, the algorithm might not be confident about its classifications. Alice then must deselect relevant events or select more events. Alice must
do this several times until the algorithm is certain about the classification. Alice also has the option of repeating the demonstration. The machine-learning algorithm was designed to be fast so Alice can get immediate feedback on the classifications.

4 Design

This section presents design principles and solutions that address the problems of allowing the efficient use of raw data (e.g., recorded sound and video) to assist the recollection of demonstrations and visualizing large amounts of data.

One principle that guided the design of the *a CAPpella* interface is parallelism (Tufte, 1997). Parallelism allows viewers to connect and compare information. This is important because users need to associate many different kinds of events. For example, ringing a doorbell has two associated information the person is near the door and the doorbell is ringing. Parallelism is also useful for long demonstrations where people change states all the time (e.g., a group meeting).

Another graphical principle used in the design of the *a CAPpella* interface is macro and micro readings of information (Tufte, 1990). Macro readings are global information that can be grasped at a glance, while, micro readings are local information that is retrieved by closer examination. By allowing macro and micro readings, users can compare events that happen far apart from each other, and look into the details of a short duration of time.

5 Paper Prototype

We conducted an informal paper prototype test to answer three questions we had during the design process:

1. What is the best visualization for the data?
2. Which do users prefer: the vertically oriented view or the horizontally oriented view?
3. In selecting relevant events, which do users think is more important: transitions (where states of events change) or intervals (streams of the same events)?

5.1 Information about the tests

Paper prototypes of two orientations each with three pairs of event-action visualizations were tested. Altogether, 6 different prototypes were tested.

The first visualization designed is the “cell” version. In this version, each event is represented as a cell and each cell contains the value of the event. This design is similar to spreadsheets.
The second version is the “strips” version. In this version, similar events are grouped together to form strips of events, thus the name. Although individual events are not delineated, users can still select events individually.

The third version is the “dots” version. This version is similar to the “cells” version, but the values are represented as dots and shapes. Numerical values are not displayed; instead, values are represented either in the quantity of dots, the sizes of the dots, and color of dots.

The six user interfaces were tested with three users (A, B, and C). The users were tested separately. The ages of the users range between 19 to 27 years old. Two of the three users are female (B and C). One of the users is an undergraduate (C), while the other two are graduate students (A and B). All users said they were very comfortable with using computers.

The six user interfaces were tested in different order for each user.

The task of the three users was to select the relevant events in a demonstration. Users selected relevant events on the paper prototype by marking them with a marker. The demonstration was about “going to lunch” and was performed and recorded with a video camera before the test. In the demonstration, there were two people, Alice and Bob. The events and actions that happened in this demonstration were:

- Bob is logged into a computer and is sitting in front of it.
- The phone rings and Bob talks on the phone.
- After several seconds, Alice knocks on the door. Bob opens the door.
- Bob sits in front of the computer and hangs up the phone.
- Alice and Bob start talking.
- Bob logs off the computer and proceeds to walk out with Alice.
- Bob forgot his backpack, so he walks across the room to get it, while Alice exits the room.
- Bob turns off the lights in the room and walks out.

5.2 Results

Two of the three users (B and C) preferred the vertical orientation, despite its problems. Their primary reason was that scrolling vertically is more familiar to them. B also mentioned how web page scrolling is vertical and horizontal scrolling in web pages is bad. The problem with the vertical orientation is that the way some of the graphs were visualized (e.g., sound level) was “weird” (User A); its values were indicated from left-
All users disliked the “dots” version, because the values were too hard to see. Users B and C preferred the “strips” version, while user A preferred the cell version. The main benefit of the “strips” version is that it is clean, but the cell version afforded delineation of events, which is important in selection. The cell version’s problems are the ambiguity of the data in the “phone on/off” feature (A-1) and the cluttered look of the “# of people” feature (B-1). These problems can be fixed by adding labels in the “phone on/off” feature and adding color-coding in “# of people feature, but these solutions risk further cluttering the grid. A combination of the cleanliness of the strip version and the event delineation of the cell version might be a better solution. Below is the initial sketch of our final design:

In observations of the users, it was inconclusive whether users selected transitions only or intervals only. Only user B selected both intervals and transitions, while User A selected intervals only, and User C selected transitions only. When asked which they thought was more important, all users answered that it depended on the kind of event that is being selected. For example, intervals are important when selecting events in a meeting because events happen in long durations. Due to this result, events in a CAPpella are visualized so that users can select both intervals and transitions.

6 User Interface

The interface is comprised of one window divided into three main panels. The top-left panel contains the player panel for playing video and audio data. The bottom-left panel displays classifications. The right panel displays the events from the demonstration and is the main interaction panel where users select and deselect events. This section will describe each component of the interface in more detail.
6.1 Log Views Panel

The log views panel contains the views that visualize the logs. Events and actions are both displayed. Currently, there are six different kinds of data displayed by a CAPpella:

- **Boolean events.** An example of a Boolean event is whether the phone is in use or not.

![Boolean event example](image)

- **Integer- and decimal-valued events.** These events are displayed as line graphs. The number of people is an integer-valued event.

![Integer event example](image)
• *Multiple-point events*. Each event is represented by multiple points (pairs of numbers). An example is the location of people at any given time.

- *Multiple-string events*. Each event is represented by a set of strings. An example is the set of action events.

- *RFID events*. Events from RFID sensors are represented as a location associated with three Boolean values. The Boolean values indicate whether the 3 preset RFID tags\(^1\) are present or not.

### 6.2 Zoom Slider

The zoom slider allows the user to get a micro and macro view of the information. In the “zoomed out” view below, the user can see that the peak at 10 seconds is the only peak for 30 seconds. Zooming into the data the user can see the many local peaks, otherwise, invisible in the “zoomed out” view.

---

\(^1\) A form of identification using Radio Frequency Identification Device (RFID).
It should be noted that changing the zoom level also changes the granularity of the cells in the log views. This means as the user zooms more out of a view, selecting a cell corresponds to multiple events.

### 6.3 Player Panel

This panel is where raw data (in this case, video) is played back. This feature highlights the use of parallelism. The video play head and the time marker are synchronized, that is, moving the play head also moves the time marker and vice versa. By synchronizing them, the user can easily relate event information with the raw data. In the following example, the user can see (at the time indicated by time marker) that there is exactly one person in the room and that the phone is in use. Since the video is synchronized with the time marker, the user can easily see that the person is also holding a piece of paper on a table.

![Figure 6. Playhead and time marker are synchronized.](image)

### 6.4 Classification Table

Classification table is used for giving feedback to the user about how the classification is improving or deteriorating. The table also displays various classifications of a demonstration.

### 7 Example Interface Interaction

Suppose a user, Alice, wants to program an application to recognize that a meeting is happening between two people in a conference room. Alice invites her friend, Bob, to help demonstrate to the *a CAPpella* system what a meeting is. Alice starts a demonstration session in the conference room that has sensors hooked up to the *a CAPpella* system. The sensors that are available in the conference room are: an overhead camera tracking the number of people and the coordinates of each person; an audio microphone tracking the sound volume in the room; a phone sensor sensing whether the phone in the conference room is in use or not; and a light switch sensor. The following are what happened in the demonstration:

- **0 s** The lights are off in the conference room and no one is inside.
- **5 s** Alice goes into the room and turns on the lights. She sits on the table in the middle of the room.
Bob goes into the room and sits with Alice.

Alice and Bob converses for four minutes.

Alice and Bob leave the room together. Alice turns off the light.

The lights are off in the conference room and no one is inside. Alice tells the system to stop recording.

After Alice demonstrates the activity, the AI system of *a CAPpella* extracts the events from the sensors. In this case, the events that are extracted and logged by the system are the number of people in the room; the x-y coordinates of the people, the sound volume, and whether the light was turned on or off.

Once the events have been extracted, Alice starts using the *a CAPpella* user interface. She clicks on the “Open session” button and selects from a dialog box the directory where the raw sensor data, event logs, and session information are stored. After a session is opened, Alice is presented with the visualization of the event logs (right pane), a video of the demonstration (top-left pane), and an initial classification by the AI system of the activity (bottom-left pane).

Alice plays the video to refresh her memory of what happened in the demonstration. After a few seconds, she sees herself walking into the room. She deems this event to be the start of relevant events in a meeting. Alice finds the red time marker synchronized with the video play head and selects (in the “Number of people” log view) the events when she entered the room, when the other person entered and when there were two people in the room. Alice continues to select relevant events in the “People’s location” log view; she selects the events when the two people are close to each other.

Alice continues to play back the video and then she sees the clip when she went up to the whiteboard. This happened for a very short time, so Alice uses the zoom slider to zoom into the data. Then, she deselects the “People’s location” event when the two people are far from each other.

Next, Alice selects the relevant actions. She selects the icon that indicates the light is turned on. Now, she is ready to send the relevant events to the machine-learning algorithm and clicks on the “Classify Session” button on the toolbar. The relevant events are stored in files. The use of the interface momentarily stops here while Alice starts the machine-learning system with the files that contain the events she has selected to be relevant.

When the machine-learning system has finished classifying the relevant events, Alice uses the *a CAPpella* user interface to see the classifications. Alice continues selecting relevant events and feeding the relevant events to the machine-learning system until the machine-learning system is confident about a classification.
8 Implementation Details

The programming language used to implement the user interface for *a CAPpella* is Java. Java was chosen as the language to develop the user interface because of its support for making graphical user interfaces with the Swing API.

The implementation of the user interface can be divided into 3 categories of classes: data, input/output, and UI component classes. The data classes are representations of the data structures in *a CAPpella*, which includes entries, logs, and sessions. The input/output classes are responsible for the reading and writing of data. The UI component classes are the implementations of the different components that appear in the user interface.

The most basic data structure in *a CAPpella* is the entry. An entry is a representation of an event at a single point in time. An entry has two values: a value representing the event (e.g. an integer, an x-y coordinate) and a value indicating whether this entry is relevant or not. There are six supported entry types: Boolean, integer, double-precision number, string, a set of points, and a list of strings. These entries are divided into three categories: entries with single values, entries with *multiple values*, and entries with *multiple entries*. An entry with multiple values contains a set of values and each value is neither relevant nor irrelevant; only the entry itself can be relevant or irrelevant. An example of an entry with multiple values is the entry that contains a set of points describing the x-y locations of a group of people. On the other hand, an entry with multiple entries contains a set of entries each of which can be relevant or irrelevant. An entry with multiple entries is only relevant when all its entries are relevant. An example of this is the entry with a list of string entries describing action events.

A log is a collection of entries with the same type. A log is a representation of events of a particular type over time. The number of entries in a log depends on the duration of the demonstration and the rate at which the sensors “sense” the environment, that is, retrieve states from the environment.

A set of logs composes a session. A session is the representation of a demonstration. It contains information about the raw data from the sensors, the start time, duration, and sample rate (frames/second) of the demonstration, and information about the logs. The number of logs in a session depends on the number of kinds of events that are retrieved from the sensors.

To facilitate the reading and writing of data structures and entries, which are stored in different formats, each data structure and entry type has a corresponding input/output class. Each data structure also has a corresponding UI component responsible for displaying them.

The different events visualized by *a CAPpella* are generalized to 4 different entry types to allow reuse of code and extensibility.

---

2 The documentation of the API is located at: http://abstract.cs.washington.edu/~ianli/pbdcap/
9 Conclusions

In this paper, we have introduced *a CAPpella*, a tool that allows end-users to build context-aware applications by demonstration and have discussed in detail its user interface. Like iCAP, *a CAPpella* expands the number of users who can build context-aware applications and puts into the hands of the end-users (the ones who will be interacting with context-aware environments the most) the task of specifying the environment’s behavior. *a CAPpella*’s contribution to this effort is that it makes more “natural” the task of creating context-aware applications.

Supporting programming-by-demonstration by using the classical models of machine learning is antithetical to *a CAPpella*’s goals, because algorithms that use this model are slow to train and require users to have plenty of knowledge on what is being classified. By using the interactive machine-learning (IML) model, *a CAPpella* reduces the task of the user to just correcting the mistakes of the learning algorithm. By applying the IML model in the domain of context-awareness, *a CAPpella* tests the robustness of the model.

The user interface applies some graphical principles that facilitate the user’s task of sifting through large amounts of data. Tests need to be performed to see how effective the user interface is in simplifying the work of the user.

The work that needs to be done with the interface include:

- *Integration of the user interface and machine-learning system*. Currently, the user interface and the machine-learning system are separate. Eventually, it will be beneficial if the user can start and stop demonstrations, manage the logs of different demonstrations and invoke the machine-learning system to classify sessions through an integrated user interface.

- *Improve visualization of classifications*. The only feedback *a CAPpella* provides to the user when it is learning the classifier is a table of values displaying the values of the classifications of the machine-learning algorithm. A better way of visualizing this information needs to be devised. One idea is to display a layer of colors on the log views that indicate the classification of the entries. The user can then easily know the classification of the whole demonstration just by looking at the predominant color. Implementing this requires that individual entries can be assigned a classification. Classification of individual entries is currently not supported.

For the whole system, the following are some things to work on:

- *Separate actions and events*. Actions and events are demonstrated together, so to associate more actions to a classification of events, a full demonstration of both the actions and events need to be performed. To allow reuse of classifications of events, actions and events should be separated. For example, suppose a user may want to associate another action (e.g., play my voicemail) to the event of entering a room. Currently, the user will make another demonstration, this time walking into the room, turning on the lights, and then playing his voicemail. If the actions
and events were separate the user will only have to demonstrate playing back the voice mail.

- **Order of events.** Order of events is sometimes important in a demonstration. For example, the event that a doorbell rang before someone opened the door might be important to the demonstration. The current machine-learning algorithm does not consider the order of events.

There are some concerns about the robustness of *a CAPpella*. If the sensors in an environment change, the classifiers that were learned by *a CAPpella* will still work, as long as the events derived from the sensors do not change. For example, currently the number of people in a room is tracked using a video camera. If an RFID sensor replaced the video camera, which can also be used to track the number of people in a room, the previous classifiers created with *a CAPpella* using the video camera will still work. On the other hand, if the environment itself changes (e.g., it was enlarged), classifiers that rely on environment-dependent events (e.g., temperature, x-y location of people) may not work.
Bibliography


