a CAPpella: Programming by Demonstration of Context-Aware Applications

Anind K. Dey[1], Raffay Hamid[2], Chris Beckmann[3], Ian Li[4], Daniel Hsu[3]

[1]Intel Research, Berkeley
Berkeley, CA USA
anind@intel-research.net

[2]College of Computing
Georgia Tech
Atlanta, GA USA
raffay@cc.gatech.edu

[3]EECS, UC-Berkeley
Berkeley, CA USA
{beckmann,dhsu@cory}
.cs.berkeley.edu

[4]University of Washington
Seattle, WA USA
ian.li@u.washington.edu

ABSTRACT

Context-aware applications are applications that implicitly take their context of use into account by adapting to changes in a user's activities and environments. No one has more intimate knowledge about these activities and environments than end-users themselves. Currently there is no support for end-users to build context-aware applications for these dynamic settings. To address this issue, we present *a CAPpella*, a programming by demonstration Context-Aware Prototyping environment intended for end-users. Users "program" their desired context-aware behavior (situation and associated action) *in situ*, without writing any code, by demonstrating it to *a CAPpella* and by annotating the relevant portions of the demonstration. Using a meeting and medicine-taking scenario, we illustrate how a user can demonstrate different behaviors to *a CAPpella*. We describe *a CAPpella*'s underlying system to explain how it supports users in building behaviors and present a study of 14 end-users to illustrate its feasibility and usability.

Categories & Subject Descriptors: H.5.2 [Information Interfaces and Presentation]: User Interfaces – graphical user interfaces, prototyping; D.1.7 [Programming Techniques]: Visual Programming; G.3. [Probability and Statistics]: time series analysis

General Terms: Human Factors, Design

Keywords: Context-aware computing, programming-by-demonstration, end-user programming, statistical machine learning

INTRODUCTION

Twelve years ago, Mark Weiser introduced the idea of ubiquitous computing or ubicomp, where computing moves off the desktop and into the environment [26]. An important component of ubicomp is context-awareness, where applications can dynamically adapt to changes in the user's activities and environments. Common context-aware applications include tour guides [1] and smart environments [2,17]. While there has been much research in context-aware computing, most of it has been focused on building infrastructures to support programmers in building applications and on the applications themselves [2,3,4,20,22], despite the tremendous value in empowering end-users to build applications.

In this paper, we describe *a CAPpella*, a system designed to empower end-users in building these types of applications. But why focus on end-users? First, end-users have more in-depth knowledge about their activities environments than any developer. Second, if only a developer can control system behavior, the user will be unable to evolve the system when her environments or activities change. Finally, in a context-aware application, most system action is based on implicitly sensed and interpreted information about the user. Therefore, the potential for designing a system that performs the wrong action and seriously annoys users is quite high. This calls for a system that can be placed in the hands of users so they can build and configure an application to do what they want when they want it.

Currently, to develop a context-aware application, developers have two options: They may create what is essentially a rule-based system composed from individual components and sensors (the "avoid intelligence" camp [3,4,20]), or they may build a recognition-based system (the "use intelligence" camp [2,22]) and focus their efforts on integrating sensed data to interpret user intent and actions.

Both of these approaches are inaccessible to end-users. The majority of research has been in the "avoid intelligence" camp, where toolkits that only support programmers have been built. These toolkits require large amounts of code to develop simple context-aware behaviors: sensed situations with associated actions. Obviously this is not realistic for most users, as they have tremendous difficulty programming a VCR or a setback thermostat (a configurable device for setting temperature for different times of day) [8]. In the "use intelligence" camp, recognizers are often handcrafted over a period of days, weeks or even months in an attempt to optimize recognition performance. It is far beyond the ability of most programmers, let alone end-users, to specify and tune the features that go into a recognizer.

Because of this, only programmers can build context-aware applications, with end-users having little control over how these applications behave. Very little emphasis has been

CHI 2004, April 24-29, 2004, Vienna, Austria

placed on empowering end-users to build their own context-aware applications. Past work has explored support for end-user control using the rule-based approach [7,11,24]. This is a valid approach, but is limited to situations where a user can be reasonably expected to come up with a static, well-specified rule in a timely fashion that accurately describes the desired context-aware behavior.

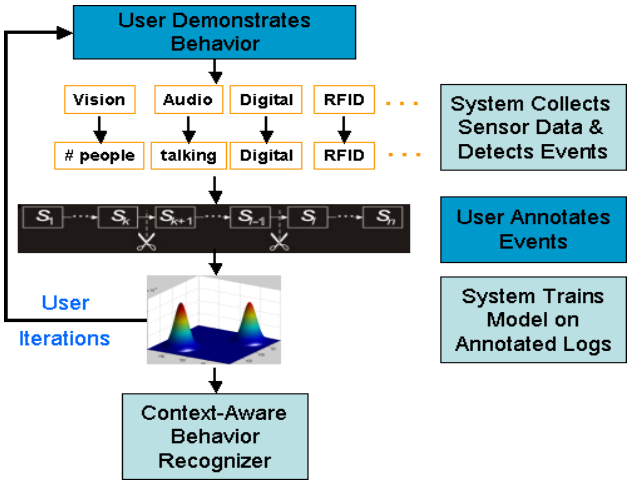

Figure 1: Design process for context-aware behaviors in *a CAPpella*. User actions shown with dark shading and system actions are shown with light shading.

This paper focuses instead on empowering users to build context-aware applications that depend on intelligence – making inferences based on sensed information about the environment. To address the issues introduced above, we present *a CAPpella*[1], a Context-Aware Prototyping environment for end-users to build applications without writing any code. *a CAPpella* uses a combination of machine learning and user input to support the building of context-aware applications through *programming by demonstration*. Specifically, a user of *a CAPpella* demonstrates a context-aware behavior that includes both a situation and an associated action (Figure 1). She uses a GUI to indicate what portions of the demonstration are relevant to the behavior and trains *a CAPpella* on this behavior over time by giving multiple examples. Once trained, she can run *a CAPpella*, and it will enact the demonstrated behavior: performing the demonstrated action whenever it detects the demonstrated situation.

Programming by demonstration (PBD) is not the only approach available to empower end-users, however we believe it offers long-term potential for supporting dynamic and complex behaviors and we investigate that potential in this paper. PBD allows end-users to build context-aware behaviors in a situated manner that would otherwise be too complex or time consuming to build. *a CAPpella* requires no writing of code and supports the building of behaviors that cannot be neatly articulated as a simple rule.

The key idea behind *a CAPpella* is that it does not require end-users to have any expertise in creating recognizers, but instead creates recognizers for them, leveraging off their natural abilities to understand their own behaviors and their ability to express those behaviors. Consider the example of a meeting. A context-aware behavior could be: when a meeting occurs, load the most recently used presentation file [25] and launch a notes-taking application. A meeting can be defined in a number of different ways, taking into account the number of people present, their location, the presence of a conversation, *etc*. Ask five people to define a meeting and you will get five different answers. But ask five people to watch a video of a meeting and, more often than not, all five will have similar insights about what features comprised the meeting and when the meeting started and ended: a classic case of recall vs. recognition. This feature specification is the essence behind *a CAPpella*.

The next section surveys previous research in the areas of context-aware computing, end user programming and programming by demonstration, providing further motivation for our work. In the following sections we present an example scenario and describe how users can use *a CAPpella* to build context-aware behaviors. We then describe the user interface and underlying machine learning system that supports end-users. We demonstrate the viability of our approach by using *a CAPpella* to learn a meeting-based behavior and a medicine-taking behavior. We also describe the results of a user study that show *a CAPpella* is a useful and usable tool. Our work with *a CAPpella* is an exploration that demonstrates the feasibility of a programming by demonstration approach for building context-aware behaviors. We conclude this paper with a discussion of the current limitations of the approach and provide future directions for this research.

RELATED WORK

Pattern recognition is a very difficult problem and there is a whole field of research that investigates how to build systems that can recognize, or classify, patterns of interest [5]. Recognizing human activity is a rapidly growing sub-field and has become more prominent in the HCI community of late [10,19,] While most research focuses on techniques for programmers to build useful recognizers, there are a number of systems that investigate the idea of putting a human "in the loop" to build pattern recognizers.

Scott *et al.* study the utility of having humans "in the loop" for optimization problems and show that this is a useful approach as long as users are focused on tasks that humans excel at, including identifying useful areas of the search space [23]. Interactive Evolutionary Computation is an optimization method where users are asked to subjectively evaluate the output of a machine learning system [14]. The

[1] *a cappella* is a musical term that means without accompaniment. We named our system *a CAPella* because it empowers the user to act without the accompaniment of a programmer.

machine learning system optimizes its learning to obtain the output preferred by the user.

Traditionally, programming by demonstration systems have focused on supporting visually-based tasks such as filing email and web browsing [15]. Although our domain is different, we were motivated by this work and its use of mixed-initiative and active learning. The mixed-initiative approach uses agents and graphical widgets to obtain input from a user, to both help a recognizer improve its recognition ability and to resolve ambiguity [9,16]. Similarly, active learning systems make queries to the user or perform experiments to gather data that are expected to maximize performance [27]. Active learners demonstrated significant decreases in the amount of data required to achieve the equivalent performance of passive learners.

Similarly, in *a CAPpella*, the user provides data to the machine learning system, focusing the learning system on her input. We show that this approach requires small amounts of time and data to produce reasonable activity recognizers. Two systems that take a similar approach to ours and have provided us with tremendous inspiration are the Neural Network House and Crayons.

The Neural Network House is an adaptive system that controls the utilities (lighting, heating, *etc*.) in a house, inferring appropriate behaviors by observing the inhabitants of the house [17]. In this work, Mozer points out that subtle statistical behavior patterns can be exploited to control the adaptive system in the house. If the system turns the lights to a particular lighting level as an occupant enters the room, the system learns about the occupant's preferences through her behaviors. If she adjusts the lighting, it learns that she prefers a different lighting level than it chose and adjusts its model of the user accordingly. While this is an interesting approach, in order to recognize complex behaviors like a meeting, additional user input beyond this reinforcement learning is required to guide the recognition system.

Crayons was another source of inspiration for *a CAPpella*. It uses interactive machine learning to allow end users to create an image classifier [6]. Users provide images to classify and then annotate the images by coloring them, indicating the areas an image classifier should look for. Fails and Olsen showed that users could very quickly create a variety of image classifiers (*e.g.*, skin and laser pointer detectors) with Crayons. The image classifiers built can be used in a camera-based application. Crayons opens up the space of image-based recognizers to end-users in the same way that *a CAPpella* opens up the space of recognizer-based context-aware applications to end-users.

While there has been work in leveraging user input to improve machine learning, there is no system that supports end users in creating recognizers of interesting context-aware behaviors. In our work, we leverage off of much of the work discussed here, using a human "in the loop" to reduce the amount of data and time required to build useful context-aware behaviors. *a CAPpella* empowers end-users to create context-aware behaviors that would otherwise require considerable programming expertise.

The next section presents an example of how one uses *a CAPpella* to build a context-aware behavior and to demonstrate the usefulness of the tool. We will revisit this example as we describe the details of *a CAPpella*.

EXAMPLE APPLICATION

Here, we describe how users interact with *a CAPpella* to create such an example behavior, a meeting. One can imagine a meeting scenario where a user is having a phone meeting and she wants to turn the lights on and launch an application to record notes from the meeting. Such a scenario is quite common and is often repeated in workplace environments. The user wants her smart environment to recognize that she is in a meeting, but cannot easily define the conditions that comprise a meeting. Current systems support the *a priori* creation of a static, and therefore likely brittle, heuristic that when true, indicates a meeting is occurring. When a meeting is recognized, they perform the desired actions. Instead, users can now use *a CAPpella* to specify context-aware behaviors *in situ*.

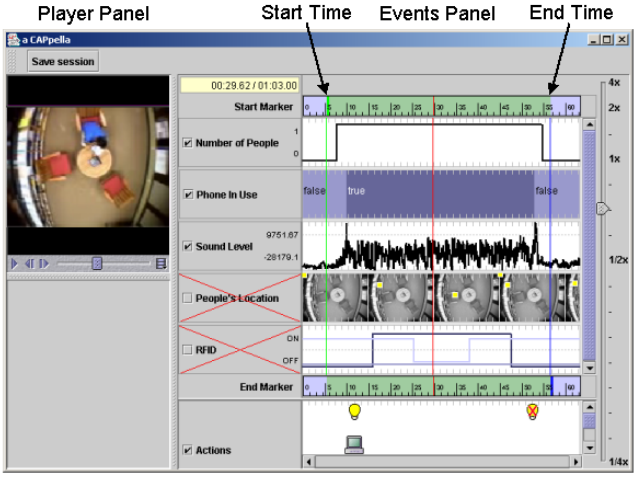

Figure 2: *a CAPpella* user interface being trained for a meeting. The user has selected a start and end time and deselected the location and RFID data streams. The actions shown are turning the lights on and off, and starting the notes recording program.

When our user is ready to create a context-aware behavior, she starts *a CAPpella*'s recording system. This captures data from all the sensors that are available to the system: video camera, microphone, radio frequency identification system (RFID), a switch that indicates whether the phone is in use, and instrumented actuators to detect actions such as logging in and logging out of a computer, sending an email, turning on or off a light, *etc*. She starts her meeting and performs the actions she would like her smart environment to perform on her behalf. When the meeting is over, she stops the recording system and uses *a CAPpella*'s user interface to view what was recorded. The user interface, shown in Figure 2, displays the data streams that were recorded and allows the user to play them back.

The user interface is divided into three parts. In the left frame, there is a video player that allows the user to view the recorded video and listen to the recorded audio. In the right frame at the top, the user can view events detected in the recorded sensor data, and on the bottom, she can view actions that she took during the recorded session. After viewing the captured data, she can *annotate* the data: selecting the streams of information she considers to be relevant to the behavior being created and the actions she wants *a CAPpella* to perform on her behalf. She sets a start and end time for all the streams to indicate when the behavior started and when it ended, as shown in Figure 2.

When she is finished pruning the data, she can train *a CAPpella* on this data. The user repeats this process a small number of times over a period of days or weeks and improves *a CAPpella*'s ability to recognize this behavior with the new data. In each subsequent iteration, the user can first test *a CAPpella*'s ability to recognize the demonstrated behavior on the newly annotated data to see how well it performs, and if necessary, train it on this additional data. If desired, she can go back and view previously trained data and re-test the updated recognizer against it. When *a CAPpella* has improved enough that it recognizes captured data correctly on a regular basis, the user tells it to constantly collect data. It then looks for the "programmed" situation (*i.e.* a meeting) in the live data, and when detected, it performs the specified actions on her behalf.

To summarize, a user first records a behavior – situation and action – that she wants *a CAPpella* to learn. She selects relevant events from the recording and uses them to train. After a sufficient number of training examples have been provided, she tells *a CAPpella* to recognize the situation, and when it does, it performs the demonstrated actions.

A CAPPELLA DESIGN

Here, we describe the design of *a CAPpella* and show how it supports programming of context-aware behaviors by demonstration. *a CAPpella* has 4 main components: a recording system, an event detection, a user interface and a machine learning system. We discuss them in detail here.

Recording System

In order for a user to demonstrate a context-aware behavior, *a CAPpella* must have multimodal sensing capability to capture both the *situation* and the *action* that should be taken. *a CAPpella* currently uses an overhead video camera, a microphone, RFID antennas and tags, and a switch that detects whether a phone is in use to capture events that occur during the demonstration of the situation. It uses an instrumented light switch, an audio alarm and an instrumented computer (for login, logout, sending email, loading recently used files and for capturing user notes) to capture events that occur during the demonstration of the action. It is easy to add additional sensors to *a CAPpella* and we have plans to do so. However, these sensors are sufficient to investigate the feasibility of our approach. When the user starts the recording, the sensors begin storing time-stamped data into separate logs, one for each sensor.

Event detection

When the user stops the recording system, the sensors stop sensing and event detection on the data logs begin. Event detection is the process of deriving higher-level events from the raw data produced by the video camera and microphone, and any other sensor that does not directly produce higher-level events. For each frame of the captured video, the number of people in the scene and their *x*-*y* location within the video are detected and added to the event log. For each sample of captured audio, the volume level and a determination of whether there is someone talking are output. Additional detection could be added, but this was sufficient to investigate our approach.

Visual events

To determine the number of people in a scene and their location, we built a simplified implementation of a multi-hypothesis framework – Bramble [13]. Bramble uses a weighted mixture of Gaussian densities to determine the likelihood that a part of the scene belongs to the foreground (*i.e.* is something that has been added to the scene) or the background (*i.e.* is something that existed in the scene before people entered it). When *a CAPpella* is installed, a background model is created (either by the installers, or by the end-users themselves using a tool we provide.) by capturing a number of short video clips containing only the background under a number of different lighting conditions (to improve robustness). Foreground models are also created for different numbers of people to improve the visual event detection system's ability to detect people and their location in captured video. When the visual event detection system examines the user's captured video data, it uses CONDENSATION, a particle-filtering framework. This is a technique that hypothesizes the existence of objects of interest in multiple locations and uses comparisons of video frames to the foreground and background model to confirm these hypotheses [12]. When all the hypotheses have been checked with respect to the observed image, the number of people in the scene and their location are estimated on the basis of the most likely hypothesis. These estimates are then written to a new timestamped log file. While we provide a tool to support end-users in creating event detectors for the number of people and their location, one could imagine using Crayons to allow them to perform this task more easily.

Audio events

To determine whether people are talking in a scene, we provide a tool that helps a user capture audio clips of ambient sound and talking. The tool takes these clips and uses a K-Means clustering analysis [5] to model these conditions. The captured audio file is compared to the two clusters (talking and not talking) to determine which of the two clusters or conditions it is closer to. The results of this comparison are output to a new timestamped log file.

User interface

a CAPpella's user interface drew inspiration from commercial tools that display and let users interact with multiple streams of information such as Apple iMovie™ and

Adobe Premiere®. The user interface was designed iteratively, starting with a paper prototype and implementing multiple versions, with evaluations at each stage. The paper prototypes tested three different representations of logged data: cell-based, where each cell showed the numerical value of the sensor; strips-based, where data with the same numerical value were grouped together; and icon-based, where icons were used to abstractly represent sensor data and color, size and density of icons were used to encode information. We also created variations to test whether users preferred to view logged data with time represented horizontally or vertically.

We gave our paper prototypes to three users and asked them to select the relevant features of a sample demonstration: a user is talking with a friend in a lab and then leaves the room for lunch. As the user leaves the room, he turns the light off. The paper prototypes represented the streams of information that a sensor-augmented environment could capture: the number of people and their locations in the scene, phone ringing, conversation over the phone between the people in the scene, the user logging in and out of his computer and turning off the lights as he leaves the room with his friend. Users were asked to select the relevant features of the scene using the prototypes.

Overall users preferred the horizontal view of the data to the vertical view. While the vertical view afforded more natural horizontal scrolling, most of the data streams were more easily read horizontally (*e.g.* audio). The users disliked the icon-based view because it represented the data too abstractly and they wanted to know the numerical values. They had mixed feelings about the strips-based and cell-based views. Two users preferred the strips-based view because it provided a cleaner interface and made it easier to locate transitions in the data. The third user preferred the cell-based view because it showed the delineation of events more clearly, which is useful for selecting multiple events.

The two views also engendered different selection behaviors. One user only chose transitions (*i.e.* events where data changed from one value to another), one user only chose intervals (*i.e.* not focusing on transitions but selecting events with the same value), and the third chose both. To support the selection of both, we combined the views, labeling transitions and delineating discrete events.

The interface is shown in Figure 2. It consists of two main panels, an events panel for viewing the captured events and a player panel for watching and listening to the captured audio and video. The events panel displays the events and actions captured and recognized by *a CAPpella*. Events are displayed at the top of the panel and actions are displayed at the bottom. Different types of events and actions are represented in different ways (Figure 2). Currently *a CAPpella* renders six different abstract types of data, based on feedback from our paper prototype users. Boolean data such as whether the phone is in use or not is rendered with colors and labels. Integer data such as the number of people in a scene, and real number data such as sound level are represented as line graphs. Multiple point events such as the location of people in a scene are represented as points on a Cartesian plane. Unlike the other data types, this data requires horizontal as well as vertical space to represent it. Because of this, the resolution at which this data is rendered is less than the other data types. Multiple string events such as actions are represented with icons – hovering the mouse over each icon reveals the string that it represents. We created a specific representation for RFID events because it did not fit one of the other categories. They are represented as multiple line graphs that can have values of on or off.

Users select streams relevant to the demonstrated behavior, by clicking on the event checkboxes. A zoom slider on the right side of the panel allows users to inspect and select events at a finer granularity (*e.g.* Figure 3).

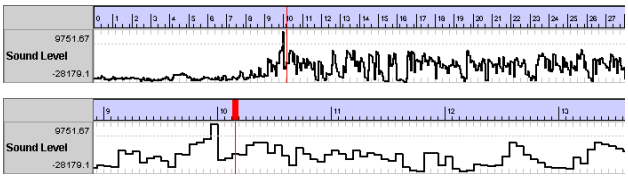

Figure 3: Zooming on event streams: zoomed out view on top and zoomed in view on bottom.

We built two versions of this interface. One supports the selection of any number of sets of events within an event stream and across event streams (*e.g.* choose time 1-5 and 10-17 for stream 1 and 2-7, 14-25 and 30-40 for stream 2). While this interface was adequate for investigating the ability of *a CAPpella* to support programming by demonstration of context-aware behaviors, informal tests with users made it clear that providing so much flexibility in selecting features may be too complicated. In our validation section, we will discuss how we used this interface ourselves to illustrate *a CAPpella*'s usefulness. A second and simpler interface we created that only allowed users to select a single start time and end time that applied to all selected streams, as was described in our initial meeting scenario. In our validation section, we will discuss a user study we performed with this version.

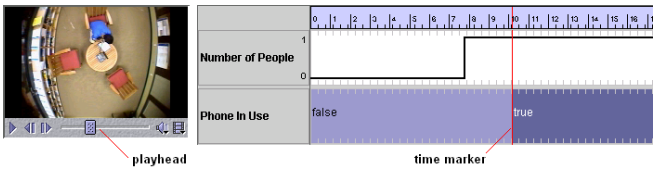

Figure 4: Synchronized playhead and time marker.

The player panel shown in Figure 4 allows a user to playback the captured video, audio and event streams (Fig. 4). The playhead in the player panel is synchronized with a time marker in the event panel, helping users to relate detected events in different streams.

Machine learning system

Once the user is done selecting the events he believes is relevant to the behavior being demonstrated, he sends this data to *a CAPpella*'s machine learning system for testing or

training. In the former case, the user indicates the name of the behavior the data should be tested against and *a CAPpella* responds with an indication of whether the behavior was recognized or not. In the latter case, the user indicates the name of the behavior to train and *a CAPpella* updates its model for this behavior with the new data.

In either case, the data being used is a collection of time series data. For its machine learning system, *a CAPpella* uses a Dynamic Bayesian Network (DBN) framework [18], a popular inference extraction stochastic framework for modeling time series data. In particular, we use a DBN equivalent of Hidden Markov Models (HMMs) [21] to support activity recognition. While HMMs are essentially a particular instance of DBNs, they do not offer the scalability or generality of DBNs. The data the user selects is combined into a single stream of observations, with an observation at a given time represented as a tuple containing data from each of the selected event streams. When disjoint time intervals are selected for different event streams, *null* data are entered for streams not selected, to indicate to the machine learning system that those event streams are to be ignored during these time intervals.

The output of a DBN is a probability distribution over its hidden states (*e.g.* whether a meeting is taking place or not). We turn this into a deterministic recognizer by creating two models from the user's input, one for the demonstrated activity (*e.g.* a meeting is occurring) and one for when the activity is not taking place (*e.g.* a meeting is not occurring). Both models are compared against test data and the model that produces the higher likelihood wins out. This comparison occurs continuously using a sliding window of 10 seconds. At every second, the system compares the output of the two models taking into account the data from the previous 10 seconds and these comparisons are filtered to remove noise. The entire process results in a short lag in producing results, which we try to optimize in future work.

For many machine learning systems, a recognizer must be hand-created by an expert for each activity that is being recognized. The key to *a CAPpella* is that the user does not need to know about or understand any of the technical details of the DBNs. All of this can be hidden from the user. Instead, with *a CAPpella,* a user can simply select the event streams and the events within them that she thinks are relevant to the behavior being trained with the DBNs. The user can create behavior models, train them and test data against them, without needing to learn the details of the models and the DBNs being used.

VALIDATION: CASE STUDIES

In the previous section, we described all the components of *a CAPpella*. In this section, we demonstrate through two case studies and a user study that *a CAPpella* can support users in programming context-aware behaviors by demonstration. The case studies illustrate how we, as designers of the system, can use *a CAPpella* to recognize two common situations, meetings and medicine taking, and perform demonstrated actions on behalf of the user. The user study shows how end-users are able to create and train effective models of a meeting using *a CAPpella*.

Case Study 1: Meeting Scenario

To investigate *a CAPpella*'s programming by demonstration approach, we tested it on the meeting scenario described in our example application. Using *a CAPpella*'s recorder system, we collected 90 samples of data: 30 of which contained 2 people having a meeting, 15 of which contained 2 people who were not having a meeting, 30 of which contained 1 person having a meeting (on the telephone) and 15 of which contained 1 person not having a meeting. Each video was between 1.5 and 5 minutes long. The videos were partially scripted as we asked the subjects in the videos to act out a meeting. The videos contained sufficient variety: different people in different locations within the scene, moving through the scene at different times, varying levels of audio and variety in the phone being used. There is additional variability added by the event detection system. We randomly chose 15 of the 2-person meeting samples and, using the *a CAPpella* interface, selected data we considered relevant to a meeting situation. For this study, we used the initial, more complex interface, which allowed us to select different time series of data from different data streams. We used this data to create and train models using *a CAPpella*.

We trained 15 different models, the first based off the first training sample, the second based off the first and second training samples, and so on. We repeated this in creating a new set of models for the 1-person meeting samples. We did this to determine the number of demonstrations needed to create a robust model with *a CAPpella*: approximately 6 (Figure 5). Training a model with a new demonstration took approximately 8 seconds with our data samples.

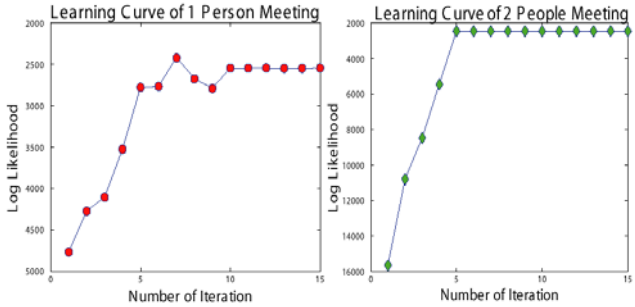

Figure 5: Learning curves for the meeting scenario models.

	1P M	1P NM	2P M	2P NM
1P M	93.3%	6.6%	0%	0%
1P NM	13.3%	86.6%	0%	0%
2P M	0%	0%	80.0%	20%
2P NM	0%	6.6%	6.6%	86.6%

Table 1. Confusion matrix of actual classification of test data for the meeting scenario models: P=person, M=meeting, NM=non-meeting.

We then tested our (stable) models against our test set: 15 2-person (2-P) meetings, 15 2-P non-meetings, 15 1-person (1-P) meetings, and 15 1-P non-meetings. The confusion matrix showing our results is in Table 1. A correct classification

occurs when the model correctly indicates that a meeting is occurring in the time interval specified by the user. The 1-P meeting recognizer recognized 1-P meetings in 93.3% of the test cases, and performed slightly worse in detecting 1-P non-meetings. The 2-P meeting recognizer recognized 2-P meetings in 80% of the test cases, performing slightly better in 2-P non-meeting cases.

Case Study 2: Medicine Taking Scenario

We tested *a CAPpella* on a second scenario, to show that it can be used with different scenarios without requiring a developer to build entirely new machine learning frameworks. In this scenario, we built a model of a user taking medicine with 5 demonstrations. If the user takes their medicine, the activity is logged into their medical journal. If they do not take their medicine by a certain time, an audio alarm goes off reminding them to take it. In this scenario, we used the same sensor modalities as in the previous scenario. RFID tags were attached to both the user and the medicine, with an RFID reader attached to the medicine cabinet. It was assumed that if the user picked up the medicine bottle, they took their medicine. In an ideal case, a weight sensor would be used to detect a change in bottle weight. The audio alarm and journal logging are PC applications, instrumented to collect interaction data.

Once again we captured a number of data samples, some with the user entering the scene and taking or not taking their medicine. While the number of people in the scene was important (*i.e.* at least one person), the RFID events were used to determine that the right user was in the scene and that it was she who picked up the medicine bottle. This is a behavior that could be built with a simple *if-then* rule, however we use this scenario to illustrate that *a CAPpella* can be used for a range of complex behaviors, without requiring any change to the underlying system. Because the situation is straightforward, *a CAPpella* is able to detect whether the user has taken his medicine with than 98% accuracy and records it to a log each time. If the user does not take his medicine, *a CAPpella* sets off an audio alarm.

Summary of Case Studies

These results show the validity of the *a CAPpella* approach. With a small number of demonstrations, we could build models that performed quite well, detecting between 80% and 93.3% of meetings taking place and close to 100% of medicine-taking situations. In the test mode, when these behaviors are recognized, *a CAPpella* controls actuators in the environment to perform actions demonstrated by the user: turning on the lights, launching the note recording program and setting off alarms and logging records.

VALIDATION: USER STUDY

To further validate *a CAPpella*'s ability to support end-users in creating context-aware behaviors by demonstration, we conducted a user study with 14 participants. In particular, we studied how effectively users were able to create models of meeting behavior in *a CAPpella*.

Our participants were not computer scientists, but had a variety of backgrounds including actors, special education teachers and students, with ages ranging from 18 to 60. We spent 5 minutes with each user in a tutorial on using the *a CAPpella* interface and describing their task. We randomly chose three 2-person meeting data samples and three 1-person data samples from the samples collected for the initial case study. We gave these to our users in random orders and asked each one to select the event streams relevant to a meeting and to select the time when the meeting began and when the meeting ended. There was no feedback provided to the user about their choices either between samples or after all samples were used.

This user study was not intended to test *a CAPpella* as a complete system. Instead, it was focused on testing end user's ability to use the interface on previously captured data to create effective models. To simplify the task and to investigate the differences between the models created with the two different interfaces, we used the constrained interface for this user study. This interface only allows users to select relevant event streams and a single start and end time. The more complex interface allows users to select relevant event streams and any number of sets of events within an event stream and across streams.

With the data from each user, we created a series of models to determine how effective users could be in creating models with the *a CAPpella* interface. We then tested the 60 video clips (15 clips each of 1-P meeting, 1-P non-meeting, 2-P meeting, 2-P non-meeting) on these models. Averaging across all 14 users, the 1-P meeting models created accurately detected 1-P meetings and 1-P non-meetings in 67.2% of the trials (std. dev. of 2.7%). Individual users' models ranged from 59.5% to 73.3% accuracy. The 2-P meeting models had a 55.5% success rate (std. dev. of 5.2%). Individual users' models ranged from 50.0% to 78.6% accuracy.

Despite the low results in the 2-person condition, the user study shows that *a CAPpella* is effective in allowing users to create models. In each of the 1-person and 2-person cases, only 3 training sets were used to create models. As we saw in our initial case study, approximately 5-6 training sets are required for a reasonable model. In the 1-person condition, the results were quite high and were improving as the number of training sets increased (correlation coefficient > 0.4), as expected. In the 2-person condition, the results were much poorer, but were still improving with the number of training sets (correlation coefficient > 0.4). Another explanation for the poorer overall data, when compared with our case study, includes the fact that the data they were viewing was not their own but was of people and locations they had never seen before, potentially causing them to be more random in the features they selected for meeting-relevance. In addition, it is possible that selecting only a start time and an end time and the appropriate event streams produces less reliable results.

The *a CAPpella* user interface appeared to have a shallow learning curve. Users did not require much assistance in using the interface during the study and their speed increased as they annotated more demonstrations. While they

suggested improvements to the interface (*e.g.* making the selection of start and end times easier), overall, users reported that the system was easy to use.

CONCLUSIONS AND FUTURE WORK

In this paper, we presented *a CAPpella*, a Context-Aware Prototyping environment. *a CAPpella* supports end-users in programming by demonstration recognition-based context-aware behaviors. It opens up the space of context-aware computing to end-users, allowing them to build *in situ* dynamic behaviors that would otherwise be too complex or time-consuming to produce in a rule-based system. Users do not need any expertise in creating recognizers. Instead, *a CAPpella* takes care of the details of creating the recognizer, using input from the user to determine what it should recognize. It allows users to demonstrate interesting behaviors a small number of times and learns from these demonstrations. A user performs a demonstration of a situation and associated action(s) and annotates the captured events, helping *a CAPpella* learn. When *a CAPpella* recognizes the demonstrated situation, it performs the demonstrated actions.

We validated the feasibility of our approach in *a CAPpella* in two ways. First, we used it to build 2 common behaviors, for a meeting and a medicine-taking scenario. We then tested *a CAPpella*'s ability to support end-users in creating models with a small number of demonstrations, with 14 end-users creating models for the meeting scenario. This study showed that users both liked the system and were able to successfully create models with it. While creating more interesting multi-person models is more difficult than single-person models, *a CAPpella*'s approach is promising.

a CAPpella is an investigation into supporting programming by demonstration of context-aware behaviors. While we were successful in this investigation, we see room for improvements and further exploration. We plan to use *a CAPpella* to build a wide variety of scenarios and to perform a more thorough evaluation with end users. We would also like to experiment with allowing users to specify more information (*e.g.* temporal ordering of events and actions) in creating behavior models. We would also like to investigate the scalability of *a CAPpella*, in supporting multiple recognizers simultaneously and in supporting multiple users in multiple locations with more sensors. The data used in this system is of much higher dimensionality than for traditional desktop uses of PBD. This requires more examples and more data be provided to our machine learning system. We are interested in improving our algorithms to reduce the number of examples required to model desired behaviors. Finally, we are interested in using the recognizers built in *a CAPpella* as part of the event detection system, This will allow users to build behaviors on top of already trained recognizers.

ACKNOWLEDGEMENTS

This work was funded in part by the National Science Foundation under grant IIS-0205644 and Intel Corporation. We would like to thank John Canny, Michael Jordan and Stuart Russell for their feedback and assistance on the machine learning aspects of this work.

REFERENCES

1. Abowd, G.D. *et al.* Cyberguide: A mobile context-aware tour guide. *ACM Wireless Networks* 3(5), 1997, 421-433.
2. Coen, M. The future of human-computer interaction or how I learned to stop worrying and love my intelligent room. *IEEE Intelligent Systems* 14(2), 1999, 8-10.
3. Cooltown homepage. http://cooltown.hp.com
4. Dey, A.K. *et al*. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *HCI Journal* 16(2-4), 2001, 97-166.
5. Duda, R., Hart, P. and Stork, D. Pattern Classification, 2nd Edition. John Wiley & Sons Press. 2001, 526 -530.
6. Fails, J.A. and Olsen, D.R. A design tool for camera-based interaction. *Proc. CHI 2003*, 449-456.
7. Gajos, K. *et al*. End user empowerment in human centered pervasive computing. *Proc. Pervasive 2002*, 134-140.
8. Gregorek, T. The energy revolution. *Electronic House* 6, 1991, 10-15.
9. Horvitz, E. Principles of mixed-initiative user interfaces. *Proc. CHI '99*. 159-166.
10. Hudson, S.E. *et al*. Predicting human interruptibility with sensors: A Wizard of Oz feasibility study. *Proc. CHI 2003*, 257-264.
11. Humble, J., *et al*. 'Playing with your bits': user-composition of ubiquitous domestic environments. *Proc. UBICOMP 2003*, to appear.
12. Isard, M. and Blake, A. CONDENSATION – conditional density propagation for visual tracking. International Journal of Computer Vision 29(1), 1998, 5-28.
13. Isard, M. and MacCormick, J. BraMBLe: A Bayesian Multiple-Blob Tracker. *PRESENCE* 8(4), 1999, 367-391.
14. Kochar, S. and Friedell, M. User control in cooperative computer-aided design. *Proc. UIST '90*, 143-151.
15. Lieberman, H. Your wish is my command: Programming by example. Morgan Kaufman. 2001.
16. Mankoff, J. *et al*. Interaction techniques for ambiguity resolution in recognition-based interfaces. *Proc. UIST 2000*, 11-20.
17. Mozer, M.C. The neural network house: An environment that adapts to its inhabitants. *Proc. AAAI Spring Symposium on Intelligent Environments 1988*, 110-114.
18. Murphy, K. Dynamic Bayesian Networks: Representation, Inference and Learning. PhD Thesis, UC Berkeley. 2002.
19. Oliver, N. *et al*. Layered representations for human activity recognition.. *Proc. International Conference on Multimodal Interfaces 2002.*
20. Pascoe, J. The Stick-e Note Architecture: Extending the interface beyond the user. *Proc. Intelligent User Interfaces 1997*, 261-264.
21. Rabiner, L.R. A tutorial on Hidden Markov Models and selected applications in speech recognition. *Proc. of the IEEE*, 77(2), 1989, 257-286.
22. Schmidt, A. and Van Laerhoven, K. How to build smart appliances. *IEEE Personal Communications* 8(4), 2001, 66-71.
23. Scott, S., *et al*. Investigating human-computer optimization. *Proc. CHI 2002*. 155-162.
24. Sohn, T. and Dey, A.K. iCAP: An Informal Tool for Interactive Prototyping of Context-Aware Applications. *Extended Abstracts of CHI 2003*, 974-5.
25. Trevor, J. *et al*. Issues in personalizing shared ubiquitous devices. *Proc. UBICOMP 2002*, 56-72.
26. Weiser, M. Computer for the 21st century. *Scientific American,* 265(3), 1991, 94-104.
27. Wolfman, S.A. *et al*. Collaborative interfaces for learning tasks: SMARTedit talks back. *Proc. IUI 2001*, 167-174.