IEEE Computational Intelligence Society
Transactions on Evolutionary Computation

XCS for Personalizing Desktop Interfaces

Journal:	*Transactions on Evolutionary Computation*
Manuscript ID:	TEVC-00120-2008.R2
Manuscript Type:	Regular Papers
Date Submitted by the Author:	20-Jan-2009
Complete List of Authors:	Shankar, Anil; University of Nevada, Reno, Dept. of Computer Science and Engineering Louis, Sushil; University of Nevada, Reno, Dept. of Computer Science and Engineering
Keywords:	Intelligent systems, Knowledge based systems, User modeling, Adaptive systems, Artificial intelligence
Note: The following files were submitted by the author for peer review, but cannot be converted to PDF. You must view these files (e.g. movies) online.	
bare_jrnl.tex	

XCS for Personalizing Desktop Interfaces

Anil Shankar and Sushil J. Louis, *Member, IEEE*

Abstract

We investigate whether *XCS*, a genetic algorithm based learning classifier system, can harness information from a user's environment to help desktop applications better personalize themselves to individual users. Specifically, we evaluate *XCS*' ability to predict user preferred actions for a calendar and a media player. Results from three real world user studies indicate that *XCS* significantly outperforms a decision-tree learner to successfully predict user preferences for these two desktop interfaces. Our results also show that removing external user-related contextual information degrades *XCS'* performance. This performance degradation emphasizes the need for desktop applications to access external contextual information to better learn user preferences. Our results highlight the potential for a learning classifier systems based approach for personalizing desktop applications to improve the quality of human-computer interaction.

Index Terms

Evolutionary Computation, Genetics-Based Machine Learning, Learning Classifier Systems, XCS, Decision-trees, User-Context

I. Introduction

XCS, a genetics based machine learning scheme, has been used for a wide variety of real world applications; few of the areas where researchers have had success include data mining, controlling traffic signals, mapping FPGA architectures, and clinical research databases [1], [2]. In our research, we use *XCS* within our user-modeling framework, **Sycophant**, for learning to predict a user's preferences for desktop computer applications.

Most desktop applications such as a media-player or calendar rely on keyboard activity, mouse usage or an internal clock to provide input (or context) for their information processing. This reliance on meager contextual information makes such interfaces only partially aware of a user and her environment. We believe that accessing additional contextual information may enable user interfaces to better adapt their actions towards individual users.

For example, if Jane prefers to turn her media player *volume down* when she is talking with someone in her office, her media player however does not know whether Jane is talking to someone, and therefore cannot adapt to Jane's preference. In a different context, if Jane prefers to *pause* her music when she leaves her desk, her media player is unable to detect Jane's absence to pause the music. Clearly, Jane's preferences for her media player changes whether she is talking with someone in her office or when she is not at her desk.

As another example, consider Jack, who unlike Jane, prefers to *pause* his media player while chatting with someone in his office; he prefers to *decrease* his media player's volume when leaving his desk. Like Jane, Jack's media player preferences are

Anil Shankar and Sushil J. Louis are with Evolutionary Computing Systems Laboratory, Department of Computer Science and Engineering, University of Nevada, Reno NV, 89557 USA e-mail: {anilk, sushil}@cse.unr.edu.

context dependent. From our media player example, we note the following about user preferences. First, a user's preference for an application action (changing volume, pausing music) depends on the context of use (talking with someone or leaving the desk), Second, application action preferences vary from user to user in the same context of use; Jane and Jack have different preferences for the same media player application while talking with someone in their office and when they are not at their desks. In addition to a user's context-dependent variation in application action preferences, the sparse, noisy, temporal, and discontinous nature of this problem domain makes learning to predict user preferences for desktop applications a challenging task.

If applications harnessed contextual information from a user's external environment they could *learn* to better predict user preferred actions and thereby improve that user's interface experience.

In this paper, we compare a learning classifier system's performance against other learners' performance in predicting user preferences of the sort outlined above for two applications. *Sycophant*, our generalized user modeling framework, gathers simple contextual information from both the internal and external environment of a user [3], [4]. A web-camera checks for movement (motion) in a user's environment and we label this contextual information as *motion*. Similarly, a microphone monitors a user's environment for the presence or absence of speech. In addition to sensing a user's external environment, we also monitor keyboard and mouse activity. *Sycophant* aggregates the external and internal contextual data from these four sensors and processes the sensors' information to extract user-related contextual features. *XCS* within *Sycophant* uses these features to generate a preference model for an individual user. An application can then leverage this user model to predict user preferred actions. For example, Jane's media player would leverage a learned user preference model to automatically pause her music whenever Jane left her desk. *Sycophant* primarily uses *XCS* to construct such an application specific preference model for an individual user.

While we briefly explain our user studies design and methodology in the subsequent sections, the first author's dissertation gives further details about our user studies [5]. This works' primary focus is twofold:

1) To demonstrate a learning classifier system's feasibility for learning to predict user preferred application actions.
2) To compare machine learning techniques for context-aware applicatons' user preference learning.

We conducted three real world user studies with **Winamp** and **Google Calendar** and our results from these studies can be summarized as follows [6], [7]: First, *XCS* successfully learned an individual's preferences for an application action. Second, *XCS* achieved the highest prediction accuracy and statistically significantly outperformed an implementation of C4.5 (decision tree) and zero-R (baseline rule learner) on these user preference prediction tasks [8]. Third, external user-related contextual information such as motion and speech better enabled XCS to predict a user-preferred action. Finally, *Sycophant* successfully context enabled two applications *Winamp*, *GoogleCalendar*, and generalized learning user preferences for multiple users for these applications. These results highlight the promise for an evolutionary learning system based approach for personalizing desktop user interfaces.

The rest of the paper is organized as follows. Section II introduces *user-context* and gives *Sycophant's* architectural details. Once we have provided this background, we review related work. Section III examines related work by researchers in context aware systems, interruption based studies, and *XCS* applications. We briefly overview *XCS*, its system operation, and describe

condensing classifiers in Section IV. Section V provides the design and procedure of our user studies and shows that *XCS* outperforms a popularly used decision tree learner for predicting user preferences. We highlight the feasibility of a classifier systems based approach to personalize desktop applications based on results in Section VI. Finally, we summarize our results and examine a few avenues for future research to use *XCS* in long term studies for personalizing user interfaces.

II. The Sycophant Framework for Personalizing Interactive User Interfaces

In this section, we first define **user-context** and then describe *Sycophant*, our context-aware user modeling framework available from the author's website based on the Open Standards software requirements license [9], [10]. *Sycophant* comprises four layers: Sensors, User-Context, Learning Services, and an Application layer. In the next two subsections we describe these four layers and the system's operation. Our software engineering paper provides usage examples of *Sycophant's* Application Programming Interface (API) [11].

A. User-Context

Researchers in ubiquitous computing have done extensive work to standardize a clear definition of *context* [12]–[14]. Dey gives one of the widely accepted definitions of context and defines context as [15]:

> Context is any information that can be used to characterize the situation of an entity. An entity is a person, place or object that is considered relevant to the interaction between a user and an application, including the user and the application themselves

We extend Dey's broad definition of context to make it applicable to a desktop environment, and define *user-context* as [3]:

> Any user-related contextual audio and video information in the vicinity of a desktop computer.

With this working definition, we structure *Sycophant*, our user modeling framework into four layers. The next section describes these layers.

B. Architecture

Figure 1 shows *Sycophant*'s four layer architecture. Context sensors in the sensors layer gather information from a user's environment and store this information in the user-context layer. From this sensor data at the user-context layer, we extract user-context features for different sensors. For example, we extract the *Count* feature to check the number of times a sensor was active in the last five minutes. Section II-B2 explains user-context features that we extract from the sensor data. The learning services API then helps to format user-context features into an appropriate data format for machine learning algorithms in the learning services layer. In this layer, a machine learner generates an application-specific user model that predicts a user's preferences. The application layer supports multiple desktop applications that harness the generated user-model in the learning services layer to predict user-preferred actions. We describe these layers in detail in the next four subsections.

1) Sensors Layer: At the lowest level we extract raw sensor information from a user's environment using a clean, well-defined sensors API. *Sycophant* currently supports four sensors: a motion sensor, a speech sensor, a keyboard sensor, and a mouse sensor. However, the sensors API is designed to easily support the addition of new sensors. We use the sensors API to

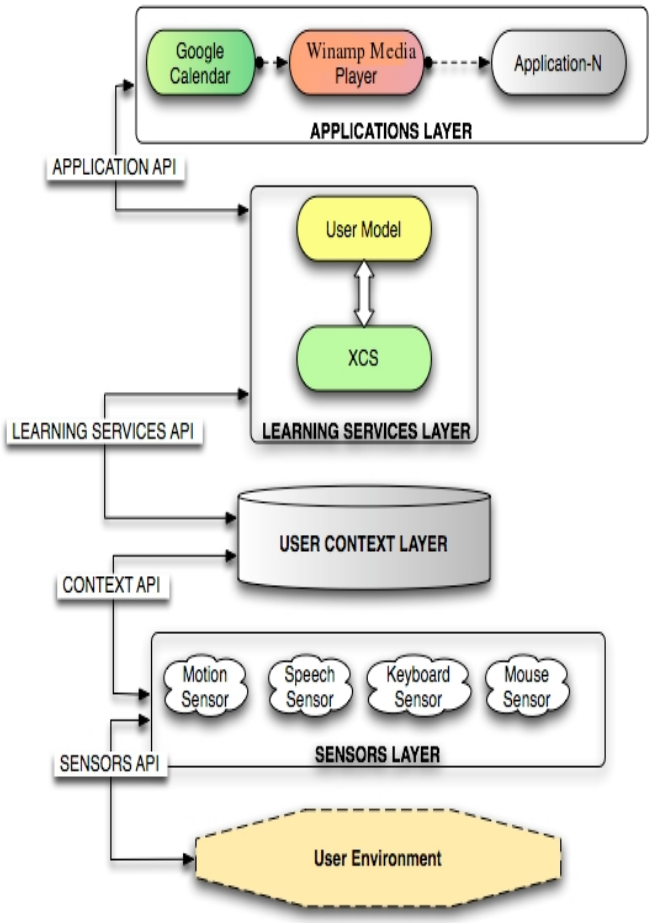

Fig. 1. Sycophant's Four Layer Architecture. The figure shows Sycophant's layers along with the APIs used to access different services in the layers.

TABLE I
USER-CONTEXT DATA EXAMPLE

Feature	Sample Value(s)
User-identifier	user-1
Motion (Any1, All1, Any5, All5, Count)	1, 0, 1, 0, 16
Speech (Any1, All1, Any5, All5, Count)	1, 0, 0, 0, 4
Keyboard (Any1, All1, Any5, All5, Count)	0, 0, 0, 0,0
Mouse (Any1, All1, Any5, All5, Count)	0, 0, 0, 0, 0

set up a sensor, associate a log file with that sensor, and start (or stop) the sensor. The next layer up, user-context layer stores this context data gathered from different sensors for use by the learning services layer.

2) User-Context Layer: We use the context API to extract user-related contextual features from the raw sensor data collected by the sensors layer. For example, we use this API to examine the motion log file to check if the motion sensor was active in the last five minutes (All5). This layer stores a number of user-related contextual features for each of the four sensors in the sensors layer. In this paper, we extract the following five user-related contextual features based on user studies conducted by Shankar et al., and Fogarty et al. [3], [4], [11], [16]:

- Any1: checks if a sensor was active in the last minute when the sensor was polled every fifteen seconds.

- All1: checks if a sensor was active in all the fifteen second intervals in the last minute.
- Any5: this is similar to Any1, except that we check sensor activity history in the last five minutes.
- All5: this feature is similar to All1 with the sensor history checked in the last five minutes.
- Count: checks the number of times the sensor was active in the last five minutes. If we poll a sensor every 15 seconds, a sensor count can have a maximum value of 20 for five minutes.

Table I shows sample values for each of these features for *Sycophant's* four user-context sensors. For the motion sensor, *Count* has a value of 16 showing that this sensor was active 16 (out of 20) times in the last five minutes. The *Any5* and *All5* values are 1 and 0 indicating that the motion sensor was active in the last five minutes when the sensor was polled every fifteen seconds but was not active during *all* of the 20 fifteen second intervals during the same period. Similarly, *Any1* is 1 for the motion sensor indicating that this sensor was active in the last minute, and *All1* is 0 showing that the motion sensor was not active during all of the four 15 second intervals during the same period. We can similarly interpret Table I for the speech sensor and note from *Count* that this sensor was active $4/20$ times in the last five minutes while being active in the last minute. The 0 values for keyboard and mouse features indicate that these sensors were not active. From these sample sensor values we notice that there was some motion and speech activity but no keyboard or mouse activity. Over time these user-context features are used by different machine learning algorithms to generate an application specific user-preference model at the learning services layer.

3) Learning Services Layer: The learning services layer hosts *XCS* and a set of other machine learning algorithms provided by *Weka's* machine learning toolkit [17]. *Weka* is an open source collection of machine learning algorithms for data mining tasks. We preprocess user-related sensor features from the context layer into an appropriate data format for a machine learning algorithm using the learning services API. We select a machine learner using this API and generate an application-specific user model that predicts user-preferred actions. For example, first we use the learning services API to select *XCS* for learning a user's preference for *Google Calendar* alarm types. Next, *Google Calendar* plugged in at the application layer harnesses this *XCS* generated user model (based on training data) to predict calendar alarm preferences for a user when new context data (a test exemplar) is available during calendar use.

4) Application Services Layer: The application services API facilitates plugging in multiple applications in the application services layer. Currently, we have plugged in *Google Calendar* and *Winamp*. *Google Calendar* harnesses a learned user preference model generated in the learning services layer for learning alarm type preferences for a particular user. Similarly, *Winamp* accesses its own learned preference model for an individual user in predicting one of its four interface actions. Thus we keep the user-context based preference model separate from an application that accesses this model to predict user-preferred actions. This separation makes *Sycophant's* framework flexible and modular.

C. System Operation

Figure 2 shows the operation of our system. *Sycophant's* four user-context sensors include a web-camera, a microphone, a keyboard, and a mouse. The web-camera detects user-movements and logs this *motion* information at the sensors layer. Similarly, the microphone detects the presence or absence of speech and logs that information. The system also checks for

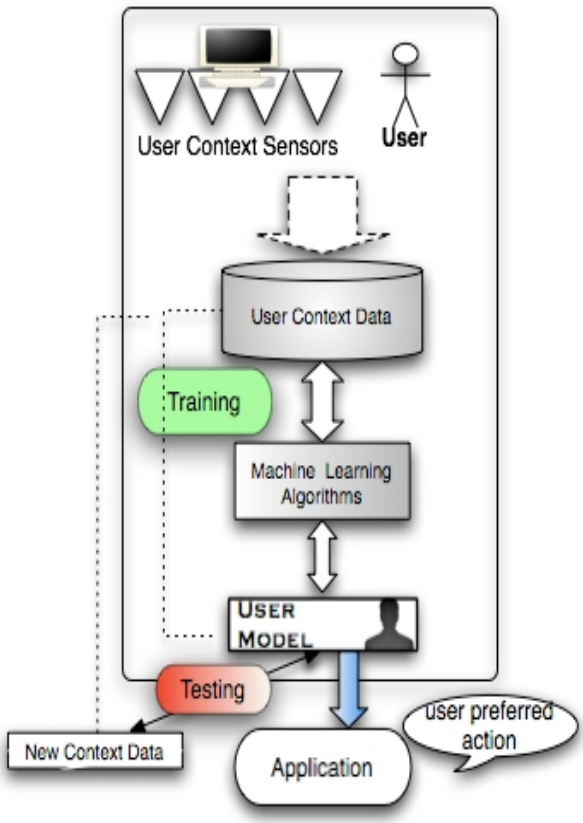

Fig. 2. *Sycophant's* System Operation

keyboard and mouse usage and logs these sensors' information as well. All our sensors operate in a binary mode; that is, if the web-camera detects motion it logs a 1 into its log file and 0 otherwise. Note that this sensor information is user-related and we consider such information as *user-context* (II-A). As explained in section II-B2, for each sensor we extract Any1, All1, Any5, All5, and Count features from the raw sensor data and store this preprocessed information in an appropriate format at the Context Layer. A machine learning algorithm, such as *XCS*, accesses this user-context data from the sensors and generates an application specific action preference model for an individual user. For example, based on Jane's *XCS* generated preference model, *Winamp* would learn to pause when she left her desk.

However we need training data to generate this preference model. *Sycophant* creates training data in the following manner: Initially, when no training data is available, *Sycophant* chooses a random application action and generates a feedback request. This feedback request asks a user whether the generated alarm was appropriate and whether the user would have preferred a different alarm. Figure 3 shows the user feedback interface. A user selects one of the interface actions in the feedback request window and *Sycophant* logs this feedback as their preference. Section V-A explains the four alarm types shown in Figure 3. We ensure that the feedback window automatically disappears after 15 seconds to mitigate the effect of annoying a user with feedback requests. A user's feedback along with the user-context sensor information is stored as a training data exemplar in the context layer. In this work, our system periodically asks the user for feedback. Before we describe *XCS* and how it leverages user-context data we briefly summarize related work in adaptive user interfaces and learning classifier systems.

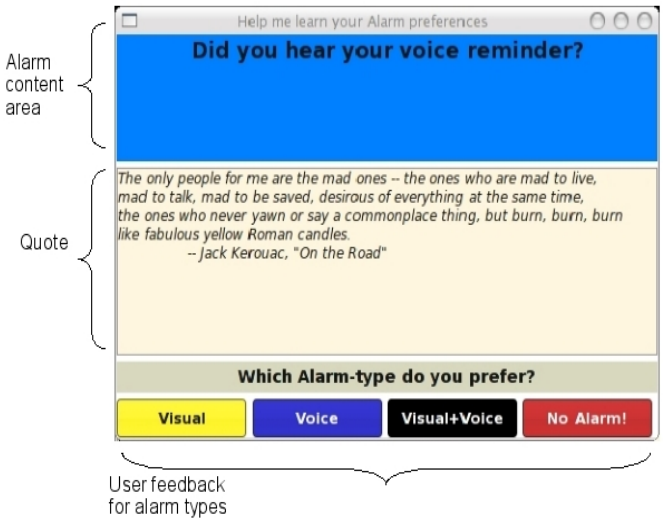

Fig. 3. A Calendar Voice Alarm. The figure shows feedback interface highlighting the alarm content area, a quote displayed as incentive for a user's feedback, and the feedback buttons for different alarm types that a user can select.

III. Related Work

In this section we synergize the advances made in context-aware systems and genetics-based machine learning research to provide a background for our *XCS*-based generalized user modeling framework.

Related research in context-aware systems has mainly addressed managing a user's attention, building statistical models to predict the state of interruptability of a user, and tracking a user's interactions with all applications for reducing a user's interaction overhead with these applications. Bailey and Adamczyk have quantitatively shown that a user's attention must be carefully managed among competing applications [18]. They necessitate such user attention management to mitigate any disruptive effects of interrupting a user at inappropriate times. Iqbal has primarily worked on managing a user's attention in multitasking environments [19], [20]. Her attention management system schedules interruptions during task executions by using a cost-benefit approach for generating interruptions. Managing a user's attention across multiple desktop applications or devices is the main focus of her research. In the *TaskTracer* project, Herlocker et al. track a user's interactions with all desktop computer applications. They track applications to organize a user's information based on tasks and make desktop applications more task-aware. TaskTracer's primary focus is to reduce overhead interaction and cognitive load while users switch tasks and improve personal productivity [21], [22]. Fogarty's *Subtle* uses sensor-based statistical models to predict the state of interruptability of a user [16], [23]. Fogarty based Subtle's design on the results of his Wizard of Oz study for building statistical models that could predict interruptability of office workers. Currently, Subtle is more suited to notebook computers and collects data from a system's opening, closing, audio analyses, mouse-clicks and WiFi sensing activities for predicting the state of interruptability of a user.

In Learning Classifier Systems (LCS) research, Kovacs has surveyed real-world LCS applications used to solve a wide variety of problems in optimization, medial domains, data mining, control and modeling [2]. *XCS*, in particular has been successfully applied in diverse domains for data mining. The next section gives *XCS'* details. Wilson used *XCS* to mine the Wisconsin Breast Cancer (WBC) data set and highlighted *XCS*' promise from both performance and pattern-discovery viewpoints [24].

XCS achieved a mean test set performance of 95 percent on a stratified 10 fold cross validation of WBC data. Saxon and Barry further tested *XCS* on the Monk's problems data set and showed that *XCS*' performance was comparable to that of neural networks and production based classifier systems [25].

We consolidate these advances in LCS research and context-aware systems. Extending interruption based studies, while Fogarty's work only showed *when* an application should interrupt a user, in addition to showing *when*, we also show *how*, that is, what type of action an application should take to generate an interruption. In contrast to research in attention management that has primarily focused on the internal environment of a desktop computer (tasks, applications), we gather user-related contextual information from both the internal (keyboard, mouse) and external (motion, speech) environment of a user's desktop computer to better personalize desktop applications. Within *Sycophant*, our context learning framework, we use *XCS*' superior predictive accuracy to enable *Google Calendar* and *Winamp* to learn an individual's preferences for different application actions. We next describe XCS and its operation.

IV. XCS

Learning Classifier Systems (LCS) were John Holland's innovative work in creating a domain-independent rule-based machine learning complex adaptive scheme [26]. Holland's work in this area laid a comprehensive foundation for Genetics-Based Machine Learning (GBML) techniques [27]. Wilson simplified Holland's LCS by removing the *bucket brigade* and the internal message list and he derived *XCS* from his ZCS and Animat programs [1], [28]–[31]. In the next four subsections, we describe *XCS*' architecture, the operation of the system when subjected to an accuracy criterion to evolve a minimal, maximally general, and accurate model for a learning task, and the parameter settings we use for learning to predict user preferences within our *Sycophant* framework [29]. The key distinction between *XCS* and a traditional LCS is that an *XCS* classifier's fitness depends on the *prediction of its expected payoff* while an LCS' classifier's fitness depends on the *actual prediction itself*. A reader familiar with XCS can skip to Section IV-D

A. XCS Architecture

Figure 4 shows *XCS'* architecture. We show *XCS* interaction with an environment via *detectors* for sensing the environmental input and *effectors* for executing an action. The environment provides *reward*, a scalar reinforcement, that is action dependent. *XCS'* core consists of a classifier population where each classifier represents a simple if-then rule; a classifier's left side consists of a single condition and its right side codes an environmental action. Figure 5 shows a classifier population. In addition to the classifier population, *XCS* components include the match-set formed for a sensed input message, the prediction array created from the match-set, an action-set corresponding to one of the actions selected from the prediction array, and a genetic algorithm that searches through the space of action-set classifiers. Before, we describe these components we briefly explain the following important classifier attributes to aid in a better understanding of *XCS*' overall operation. Wilson's papers give additional details about classifier components [1], [28], [31]. Each *XCS* classifier consists of the following important attributes:

- **Condition**: is defined over the ternary alphabet $\{0, 1, \#\}^L$, where L is a classifier's individual bit-string length. The meta character $\#$ matches either a 0 or 1. Whenever the system senses an input from the external environment, *XCS*'

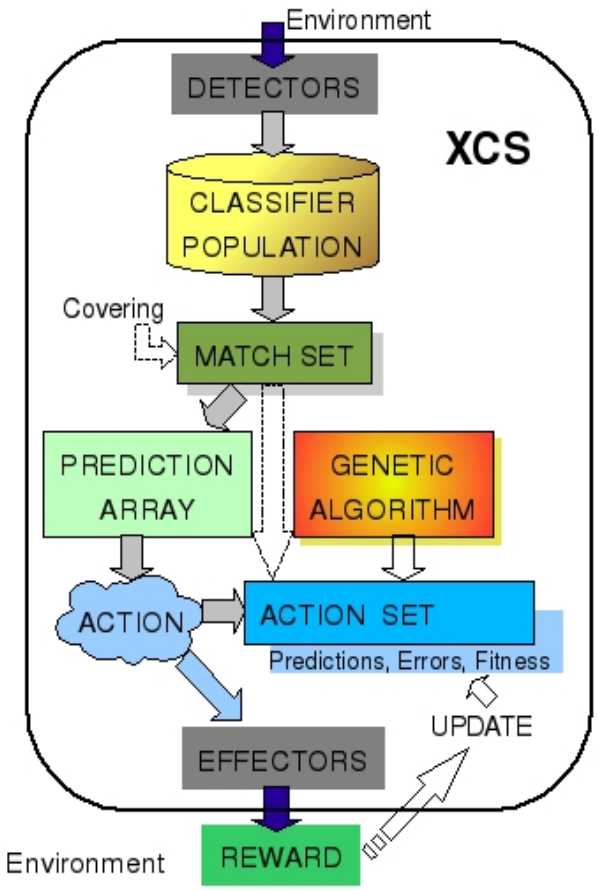

Fig. 4. The XCS Classifier System

classifiers try to match the input's condition part. For example, when *XCS* learns to predict a user's *Google Calendar*'s alarm preference, the condition component corresponds to an exemplar from the user-context data. Table I shows sample user-context data values and the corresponding exemplar is:

participant-1,1010(16),1000(4),0000(0),0000(0)

A system generated classifier's condition component for the above exemplar would be as shown below. Section IV-D explains this binarization in detail.

0001,101#(10000)1000(00100)000#(00000)00#0(00000)

- **Action**: Action is an operation which a classifier can execute; generally, action is chosen from a set of finite classes. For example, in case of *Google Calendar*, a classifier's action could be a voice-alarm, a visual-alarm, both voice and visual alarms, or none. Section V-A explains these alarms in detail.
- **Prediction Estimate**: is the value of an estimated pay-off if a classifier's condition part matches the environment's input and the system executes this classifier's action. It is this key attribute which distinguishes *XCS* from a traditional LCS.
- **Prediction error**: is an estimate of error made by a classifier's predictions. The system computes prediction error after it receives an environmental reward.

- **Fitness**: represents a classifier's fitness value. In *XCS*, a classifier's fitness is calculated by reestimating its attributes if that classifier is present in the action-set. The system receives an environmental reward as a result of a classifier executing a particular action. This reward influences fitness reestimation. We describe action-set formation in the next subsection. A classifier's reward also factors in the system's predictions and errors updates. Subsequently, the system estimates accuracy and converts this estimate into a relative accuracy value. This relative accuracy value along with the learning rate β is used to update a classifier's fitness. We do not modify XCS' fitness computation mechanism but use Butz' implementation [32]; hence we refer an interested reader to Wilson's papers that give mathematical details about these intricate computations [1], [31].

We explain the other main XCS components including the match-set, the action-set and the GA in the context of system operation in the next subsection.

B. XCS System Operation

Figure 5 shows the system's operation during training and testing phases. We first give a brief overview of the training and testing phases of the system operation and next explain these phases in detail. During the *training phase*, *XCS* evolves a classifier set for a training fold (a subset of all data collected for a user) and stores these classifiers after our experimentally determined stopping criterion of repeatedly sampling 20000 random problems (exemplars) or when the training performance reaches 1.0. Thus, from the training data *XCS* learns a classification model of the target concept, that is, the interface action type to predict. During the *testing phase*, we test these training-set evolved classifiers on the testing fold (mutually exclusive subset of the training fold) and record the system performance. Thus the testing fold helps to evaluate *XCS*' predictive accuracy on unseen cases. Even though our learning set up is set up in this batch-mode, we could progressively update the training model as new data is available to the system. Such an update could either be done when Sycophant is inactive (no user-preferred actions to predict), or periodically (say, every twelve hours).

During system training a user-context data exemplar serves as the environmental input message string to the system. *XCS'* classifiers whose condition component match this input message string are grouped to form the match-set (M). For each of the actions present in M the system computes a fitness-weighted average for each of M's classifiers. The system next selects the *best-action* (experimentally determined) which is the action associated with the highest fitness in the prediction array. *XCS'* effectors send this action to the external environment. Based on this action the system receives an appropriate reward from the external external environment. This cycle of environmental input sensing, action selection and factoring in the reward is continued until the system meets the training-phase termination condition of randomly sampling 20000 problems or achieving a training-set prediction accuracy of 100 percent (experimentally determined). We store the classifiers (rules) at the end of the training phase.

During system testing a new user-context exemplar whose action component is unknown is available to the system. *XCS* uses the training-phase evolved classifiers to match the condition component of this test-exemplar. Similar to the training phase the system forms a match-set for the test-exemplar and predicts an action to execute.

In more detail, during the training phase the system forms the match-set for a sensed input message string from the

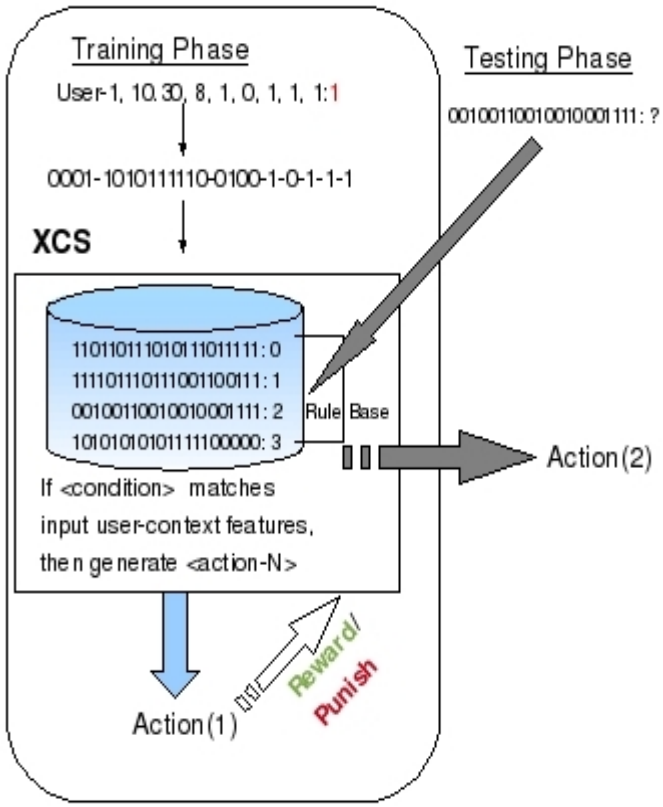

Fig. 5. XCS System Operation. The figure shows a trained XCS predicting both visual and voice alarm (action type-2) for a test exemplar.

environment. If none of the classifiers in the population match the condition component of the input message string, the system creates new classifiers with each of the possible environmental actions to form the match-set. This procedure is called *covering*. In our set-up we start with an empty classifier population and initiate covering only at system initialization.

After the system creates M, it calculates a fitness-weighted average of each of M's classifiers for each of the actions present in M. There are many action selection mechanisms, for our system's optimal performance we use the best-action selection mechanism for choosing one of M's actions (a). The system sends this action, a, to the external environment for the sensed input message string. *XCS* gets 1000 (experimentally determined) reward if it takes the correct action and 0 otherwise. Due to project time constraints, we did not investigate the theoretical basis for the success of the best-action selection mechanism, but we believe that it would be an interesting avenue for future research. At the same time, the match-set classifiers which proposed a are grouped to form the action-set (A). The action-set classifier attributes (including fitness) get updated based on the actual reward received from the environment. We run the GA on the action-set based on θ_{GA}, a threshold parameter to improve our system performance.

When the system executes a selected action and receives an appropriate reward from the environment, classifiers in the action-set whose action was chosen get their fitness values updated. This procedure ensures that accurately predicting classifiers tend to reproduce more due to their increased fitness for choosing the correct action, while inaccurately predicting low-fitness classifiers tend to get weeded out of the population. This cycle of environmental input sensing, action selection, factoring in the reward

and running a GA on the action-set classifiers is continued until a termination condition is met. Based on experimental runs, we use a termination condition of randomly sampling 20000 problems (context data exemplars from the training set) or a training set performance prediction accuracy of 100 percent. We also use *condensation* to extract a minimal classifier subset to represent the final solution for predicting user-preferred interfaces actions.

C. Condensation

Condensation consists of running *XCS* with crossover and mutation rates set to zero. This process suspends genetic search since no new classifiers are generated. However, *XCS*' selection and deletion processes continue to operate whenever the GA gets triggered. As a result, there is a tendency for less fit, less general classifiers to be weeded out of the population. Reducing the number of classifiers enables the system to extract a minimal classifier subset capable of representing the final solution [1], [29]. Our system enables condensation after our stopping criterion is met. Based on trials to tune our system performance, we choose to stop condensation after sampling an additional 20000 problems from the training set. In the next subsection, we give the value of various *XCS* parameters that we use in our system.

D. XCS' Encoding and Parameter Settings

XCS' parameter values are based on our attempts to optimally tune the system performance (test-set prediction accuracy). We start with an maximum population size of 20000. For an *XCS* classifier, β is the learning rate for updating predictions, prediction error, fitness and the action set size estimate. We set β to 0.2. The threshold for applying the GA in an action set, θ_{GA}, to optimally tune our system performance is set to 25. The GA has probability of crossover(χ) set to 0.8, mutation probability(μ) set to 0.04, and meta character (#) probability set to 0.33. The other parameter values are the default values provided by Butz [32].

A user-context data exemplar is a shown below:

participant-1, 1010(22), 0000(0), 1010(26), 1010(2):3

where the first attribute denotes a study participant. The next five are motion attributes - Any1, All1, Any5, All5, Count5 (shown in parentheses); Section II-B2 describes each of these user-context features. Similarly, the next fifteen attributes in groups of five relate to speech, keyboard, and mouse activity. The value after ":" denotes the class to which this exemplar belongs. Here, we show an exemplar for *Google Calendar* alarms, where class 3 corresponds to both voice and visual alarms. We binarize this user-context data exemplar into an *XCS* suitable format as shown below:

Binary Equivalent:

0001,1010(10110),0000(00000),1010(11010),1010(00010):3

XCS Classifier:

00011010101100000000001010110101010000010:3

This converted binary string represents one of *XCS*' classifiers in the population. The binary values before the ":" denote the *condition* part and the 3 denotes this particular classifier's *action* part. In the next section we give the design of three real world

user-studies which we conducted to evaluate the feasibility of an *XCS*-based approach for predicting user-preferred application actions.

V. User Studies Methodology

We conducted three user studies to investigate whether *Sycophant* could accurately predict an application's (*Google Calendar*/*Winamp*) action for an individual user. The Office of Human Research Protection at our university approved and validated our studies that involved human-subjects. Our research questions were refined by building upon the results in successive studies. To illustrate *Sycophant's* gradual progress in predicting user-preferences, we first present the study designs and next give the results in the next section.

Our first study was a pilot study to evaluate the feasibility of learning a user's interface action preferences based on user-context. In our second study, we examined if *Google Calendar*'s alarm action preference learning generalized across different users. In the third and final study, we evaluated if *Sycophant* could enable an additional interface, *Winamp*, and thus generalize our user-context learning approach across two widely-used desktop applications. We next describe our three study procedures in detail.

A. Pilot Study

We developed a functional calendaring application that had options for adding (or deleting) appointments for this study. Our calendar generated four types of alarms:

1) None: No alarm was generated to interrupt a user.
2) Visual Alarm: A popup window displayed the appointment text.
3) Voice Alarm: A text-to-speech synthesizer voiced out the alarm text.
4) Both: Combined visual and voice alarms.

In this study we investigated whether *Sycophant* could successfully leverage user-context information to predict one of the four calendar alarm types. Since the focus of this study was to validate the feasibility of learning to predict user preferred application actions, we recruited only three participants for this study. However we used at least ten participants in our subsequent user studies based on encouraging results from our pilot study.

Three users set appointments for their daily activities over a period of six to eight weeks. During this period we collected 323, 347, and 354 exemplars from these three participants, respectively. These three participants were graduate students in our department. We considered the task of predicting whether or not to interrupt a user with an alarm as the *Two-Class Problem*, and the task of predicting an alarm from one of the four alarm types as the *Four-Class Problem*. The two-class problem prediction accuracy helped us evaluate *Sycophant's* performance on deciding *when* to interrupt a user for comparing our results with Fogarty's interruption based studies [16]. We used the four-class problem prediction accuracy to evaluate *Sycophant's* performance on deciding *how* to interrupt a user. Figure 3 shown earlier depicts a Voice Alarm; in this figure, note that we display an interesting quote along with the alarm as an incentive for a user to provide feedback. In the next phase we plugged in *Google Calendar* into *Sycophant's* application layer. Similar to our pilot study's functional calendar, *Google*

Calendar generated the four alarm types for different study participants. Detailed results in Section V-A show that *XCS* helped *Sycophant* to accurately predict a user's preferred alarm-type.

B. Short-term Study1, Google Calendar

Figure 6 shows the user-context sensor positions for our study enabled desktop computer in our first short-term study. Section II-C explains our approach to gather motion, speech, keyboard and mouse information. In contrast with our pilot study where we collected calendar usage data from three participants over a span of three weeks, this study was a short-term study that simulated an average work day in our research lab where students read research papers. When reading papers, most students in our lab listen to music while they are sometimes interrupted with conversations from neighbors. We investigated if contextual information from these students' environment could help *Sycophant* to accurately predict their preferred alarm-types. Ten participants who were graduate and undergraduate students in our department attended four separate 45 minute sessions

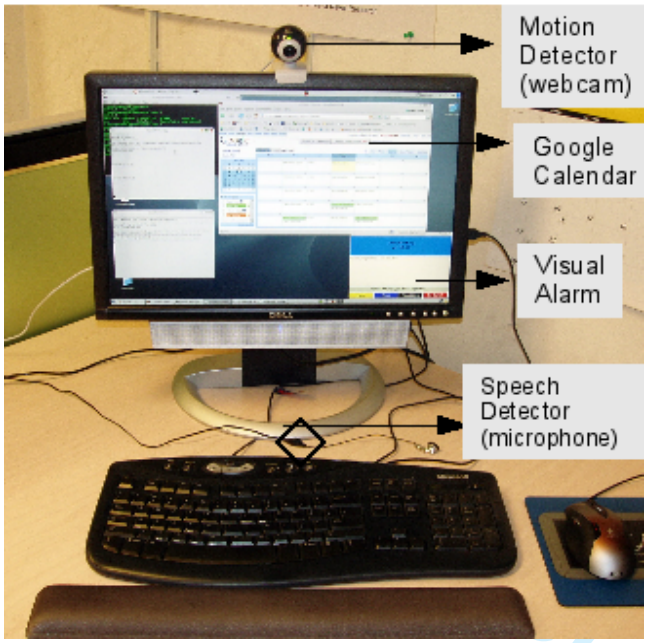

Fig. 6. Short-Term User Study Computer Setup. The figure highlights the motion sensor (web-camera), the speech sensor (microphone), and a visual-alarm generated by Google Calendar.

each. In each session, a participant read an article on the study desktop for the first 30 minutes, and answered article related questions in the last 15 minutes. *Sycophant* generated alarms for a participant during the first 30 minutes of each session. The alarm content was either study related or related to a session article. A participant provided feedback by selecting one of her preferred alarm-types whenever an alarm was generated. We collected an average of 60 user-context exemplars from our participants in the four study sessions.

Table II shows the experimental design for four study sessions. Column I shows the session numbers. The article reading lengths are shown in Column II. We show the randomized alarm order in Column III. We used this randomized design to control study variation, and the same randomized alarm order was used for all the participants in our study. The alarms $0, 1, 2$ and 3 correspond to no-alarm, visual alarm, voice alarm and both visual and voice, respectively. Column IV shows the four study conditions, that is the treatments which we applied to all our study participants. We applied Talk, Music, No-Music,

TABLE II
STUDY-1: GOOGLE CALENDAR SHORT-TERM STUDY'S EXPERIMENTAL DESIGN

I Session	II Task	III Alarm Order	IV Conditions
1	short article	0, 2, 3, 1 3,1,2, 0 1, 0, 2, 3 2, 1, 0, 3	Talk, No-music Music, No-talk No-music, No-talk Music, Talk
2	long article	0,2,3,1 1, 2, 0, 3 1, 3, 2, 0 3, 0, 2, 1	Talk, No-music Music, No-talk No-music, No-talk Music, Talk
3	short article	1, 3, 0, 2 2, 3, 1, 0 2, 0, 3, 1 3, 2, 0, 1	Talk, No-music Music, No-talk No-music, No-talk Music, Talk
4	long article	0, 3, 2, 1 1, 0, 2, 3 1, 0, 2, 3 3, 0, 1, 2	Talk, No-music Music, No-talk No-music, No-talk Music, Talk

and No-Talk as treatments. In this study, we used scripted talk to initiate conversation with a study participant and labelled this treatment as *Talk*. We played music (selected through participant survey) and we labelled this as the *Music* treatment. Suspending conversation was labelled *No-Talk* and stopping music was labelled *No-Music*.

A self-report was a participant preferred alarm type, and the construct we measured was the calendar generated alarm-type. *Sycophant* randomly generated alarms as shown in Column III. A participant read a short article in the first session, a longer article in the second session, an article of similar length in the third session and finally a short article in the last session. A subsequent participant read a long, short, short, and long articles in the study. We thus counter balanced the article lengths for every participant pair to reduce the chances of article length order affecting alarm-type preferences. Results in Section VI-B show that user-context helped *XCS* to better participant's preferred alarm types and that this user preference prediction generalized across multiple participants. In our next study, we evaluated whether *Sycophant* could context-enable *Winamp* and predict a user's preference for one of its four interface actions.

C. Short-term Study2, Winamp

In our second short-term study, we investigated if we could leverage user-context to predict a user's *Winamp* preferences. We checked if *Sycophant* could context-enable *Winamp* in addition to *Google Calendar* and thereby generalize our user-context learning approach to another desktop application. Similar to our first short-term study, 10 graduate and undergraduate students in our department participated in this study. Our study simulated computer science graduate students reading an article, listening to music on *Winamp* while having short conversations or leaving their desk for various reasons.

Table III shows the experimental design for four study sessions. *Winamp* action was the independent variable in our study.

Column I shows the session number. We show the session article lengths in II, and the treatments applied to all study

TABLE III
STUDY-2: WINAMP SHORT-TERM STUDY'S EXPERIMENTAL DESIGN

I Session	II TASK	III Conditions
1	short article	No Talk, No Map Reading No MapReading, Talk Map Reading, Talk MapReading, No Talk
2	long article	No Talk, No Map Reading No MapReading, Talk Map Reading, Talk MapReading, No Talk
3	long article	No Talk, No Map Reading No MapReading, Talk Map Reading, Talk MapReading, No Talk
4	short article	No Talk, No MapReading No MapReading, Talk Map Reading, Talk MapReading, No Talk

participants in IV. The *Talk* treatment was the same as in our previous short-term study. We again counter-balanced the article lengths for every participant pair and applied the same treatments to all the 10 participants.

Participants anonymously completed a survey questionnaire that listed a song and indicated their song preference based on a five point *Likert* scale (strongly dislike, dislike, neutral, prefer, and strongly prefer) [33]. After the 10 participants completed our survey, we chose songs that all the participants preferred at the neutral level or higher. Participants listened to these songs during their four study sessions. We kept the article reading and question answering session times the same as our first short-term study.

Each participant attended four separate sessions and each session lasted 45 minutes. While a participant read an article, she was interrupted with talk (using a script) and/or made to leave the study area to perform a place-location task on a map. Participants left their desk to find a place on a map of Yosemite National Park. We label this as the *MapReading* treatment (Column III) in our design. During these interruptions, we hypothesized that different participants tend to have different preferences for *Winamp*. To capture these preferences, computer generated requests periodically (every 70 seconds) solicited participant feedback for one of *Winamp*'s four actions (pause a song, play a song, increase volume by 10 percent, and decrease volume by 10 percent). We used a feedback request similar to the one shown in Figure 3 for recording a participant's feedback. *Sycophant* collected an average of 60 user-context exemplars (user feedback) for every participant during the four study sessions. Instead of selecting one of the four alarm types, a participant selected one of *Winamp's* four interface actions listed in the parentheses above. During the last 15 minutes of a 45 minute study session, a participant answered questions

related to the article without being interrupted. We recorded a participant's *Winamp* action preference, motion activity, speech activity, keyboard activity, and mouse activity as contextual information from their desktop environment. Our goal was to find a mapping from this contextual information to a participant's *Winamp* action preference. In the next section, we present results from these three user-studies.

VI. Results

We used three machine learning schemes for these user studies: *Zero-R*, *J48*, and *XCS*. Zero-R is a majority voting learning scheme that predicts the majority class in any data set. For example, if a participant's data had *play* in 8 out of 10 cases for *Winamp*, then Zero-R predicted *play* as *Winamp's* default action for this user. J48 is a decision-tree implementation of Quinlan's C4.5 learner in *Weka*, a popular machine learning tool kit [8], [17]. At the end of this section we discuss some of the insights we gained into using *XCS* for predicting user preferred actions and the relevance of user-context for successfully personalizing applications to individual users.

We categorize the results from our user studies to answer the following questions:

1) Can *Sycophant* accurately predict a user-preferred alarm type based on user-context?
2) Is *XCS* a feasible approach for learning user-preferences for alarm types?
3) Does user-context help *XCS* to better predict a participant's alarm type preference?
4) Does our user-context based approach to learn participant preferences generalize across multiple participants?
5) Can *Sycophant* support *Winamp* in addition to *Google Calendar*?, and can *XCS* successfully predict a participant preferred *Winamp* action?

We use a two sample t-test with a confidence interval of 95 percent to compare *XCS*' performance with that of a decision-tree machine learning algorithm [34]. We record *XCS*' performance by noting the percentage of correctly solved problems in the last 50 sampled problems. Our system assumes that a problem is solved if a classifier correctly predicts the interface action type to take for the classifier's condition component. The user-context data collected for an individual was not sufficient to create 10 cross-validation folds for evaluating a machine learning algorithm's test-set prediction accuracy. Hence we use a three-fold cross-validation approach but conduct 10 runs for each pair of training-testing folds to get a robust statistical estimate. We repeat the same procedure for the decision-tree learner, J48. Based on experiment runs to optimally tune *XCS*, we train *XCS* by making it evolve a classifier set on a training fold and store these classifiers after our stopping criterion of repeatedly sampling 20000 random problems (exemplars) or when the training performance reaches 1.0. Thus, from the training data *XCS* learns a classification model of the *target concept*, that is, the interface action type to predict. We test these training-set evolved classifiers on the testing fold and record the system performance. The testing fold thus helps us to evaluate *XCS*' predictive accuracy on unseen cases.

A. Pilot study results

Our feasibility study paper gives results of our pilot study in more detail [35]. The first author provides a more comprehensive treatment of feature design including the predictive power of each feature in his dissertation [5]. We briefly summarize this

study's results here to highlight our gradual evaluation of XCS' feasibility to predict user-preferred interface actions. For three study participants on the two-class problem of learning to predict whether or not to generate an alarm, *XCS*, J48 and Zero-R had average prediction accuracies (performance) of 89, 84 and 72 percent, respectively. Thus *XCS* and J48 performed better than Zero-R's base rate performance. *XCS*' performance of 74 and 100 percent matched that of J48 whose prediction accuracies were 74 and 100 percent for two participants. *XCS* with a predictive accuracy of 93 percent also outperformed J48 which had a predictive accuracy of 78 percent for the third participant.

On the four-class problem of learning to predict one of the four alarm types (none, visual, voice, and both), again, *XCS* and J48 performed better than Zero-R's base rate performance. XCS, J48 and Zero-R had average prediction accuracies of 97, 81 and 72 percent, respectively. *XCS* significantly outperformed J48 for two participants with predictive performance of 100 percent (for both participants) when compared to J48's 70 and 72 percent for these two participants. Also *XCS'* performance (91 percent) was worse than J48 (99 percent) for the last participant. When we examined this participant's data, we found that visual and voice alarms were preferred by this participant for $349/352$ instances. We attribute *XCS*' lower predictive accuracy to this participant's static alarm type preference; this also explained the majority voting Zero-R's high predictive accuracy.

Our pilot study results indicated that *Sycophant* could accurately predict a participant-preferred alarm type by relying on user-context, thus answering the first question which we posed in Subsection VI. Also, our two-class problem's predictive accuracy (deciding whether or not to interrupt a participant) was comparable to that of Fogarty's interruption based studies [16]. In our next study we investigated if we could learn alarm type preferences for multiple participants.

B. *Short-term study1, Google Calendar, results*

Tables IV and V show the test set predictive accuracy of Zero-R, J48, and *XCS* on the two-class and four-class *Google Calendar* alarm problems, respectively. In both the tables, Column I lists the participant, and Column II shows Zero-R's test-set prediction accuracy. Columns III and IV show J48's prediction accuracy with and without user-context features, respectively. Similarly, Columns V and VI show *XCS*' prediction accuracy with user-context and without user-context. Column VII is the statistical inference comparing *XCS*' performance with that of J48 on the alarm prediction tasks. The three numbers in

TABLE IV
SHORT-TERM STUDY1: TEST SET PERFORMANCE ON THE TWO-CLASS ALARM PROBLEM FOR GOOGLE CALENDAR.

Learning Algorithm Participant ↓	II Zero-R	III J48^{+uc}	IV J48^{-uc}	V XCS^{+uc}	VI XCS^{-uc}	VII XCS better than J48? (V > III)
1	0.5806	0.8871	**0.8548**	0.9530	**0.9015**	TRUE
2	0.5000	0.7407	0.7778	1.0000	**0.9608**	TRUE
3	0.7121	1.0000	1.0000	1.0000	1.0000	SAME
4	0.6200	0.9400	0.9400	1.0000	**0.9213**	TRUE
5	0.5455	0.8485	0.8485	0.8788	**0.8030**	TRUE
6	0.7143	0.8889	0.8889	0.9008	**0.8349**	TRUE
7	0.7460	0.9841	**0.9048**	1.0000	**0.9833**	TRUE
8	0.5625	0.7083	**0.6875**	0.8915	**0.8301**	TRUE
9	0.8163	0.9184	**0.8980**	1.0000	**0.9792**	TRUE
10	0.5246	0.6885	0.7377	0.9153	**0.7025**	TRUE
			4		9	9

the last row indicate the number of participants for whom removing user-context degraded J48's performance, the number of participants for whom removing user-context resulted in *XCS*' performance degradation, and the number of participants for whom *XCS* outperformed J48 on the alarm prediction tasks.

TABLE V
SHORT-TERM STUDY1: TEST SET PERFORMANCE ON THE FOUR-CLASS ALARM PROBLEM FOR GOOGLE CALENDAR.

Learning Algorithm Participant ↓	II Zero-R	III J48^{+uc}	IV J48^{-uc}	V XCS^{+uc}	VI XCS^{-uc}	VII XCS better than J48? (V > III)
1	0.5806	0.6129	**0.5806**	0.9697	**0.9182**	TRUE
2	0.5000	0.5185	0.5926	1.0000	**0.9227**	TRUE
3	0.7121	0.7347	0.7755	1.0000	1.0000	SAME
4	0.6200	0.7400	**0.6000**	1.0000	**0.9236**	TRUE
5	0.5455	0.5152	0.5303	0.7879	**0.4545**	TRUE
6	0.7143	0.6825	0.7143	0.7047	**0.6548**	TRUE
7	0.7460	0.9683	**0.8889**	1.0000	**0.9167**	TRUE
8	0.5625	0.5625	0.5625	0.9556	**0.9163**	TRUE
9	0.8163	0.6818	0.7121	0.9702	**0.9563**	TRUE
10	0.5246	0.5574	**0.4754**	0.8833	**0.5072**	TRUE
			4		9	10

On the two-class problem, both J48 and *XCS* perform better than Zero-R's base rate performance. Removing user-context degrades J48's prediction accuracy for 4 participants - 1, 7, 8, 9 (Column IV bold-faced values). This clearly shows that external user-context (motion and speech) helps J48 to better learn these participant's preferences. *XCS*' performance significantly degrades without user-context for all participants except for one participant. When we examined this participant's data, we found that this individual had no varying preferences and hence presence (or absence) of user-context had no effect on learning this participant's alarm type preferences. Also, *XCS* significantly outperforms J48 for 9 participants who did have varying preferences.

We notice a similar behavior of the machine learning schemes on the four-class problem of predicting one of the four calendar alarm types. Again, removing user-context degrades J48 (participants:1, 4, 7, 10) and *XCS*' (all participants except 3) alarm prediction accuracy. We ranked J48's context-features for all the 10 participants using the *Information-Gain Ratio* (IGR), a measure for evaluating the relevance of an attribute for a decision-tree, in *Weka*. In the top 15 ranked features, participants 1, 4, 7, and 10 had more user-context features (related to motion or speech) when compared to the other participants. That is, IGR showed that user-context features were important for these participants and hence removing user-context degraded J48's prediction accuracy on the two-class problem. Also we note that *XCS* outperforms J48 for all the 10 participants.

The results in this study further substantiates our hypothesis of relying on user-context to better predict a participant's alarm type preference. *XCS*' superior predictive performance across all participants (when compared to the decision-tree learner) demonstrates that it is viable technique for predicting application action preferences. This result answers the questions (2 and 4) that we posed earlier in Section VI related to *XCS*' feasibility for predicting user preferences and generalizing this preference learning across multiple participants. The performance degradation of both J48 and *XCS* highlights the importance of harnessing external user-related contextual information from a participant's environment to learn their preferences for different alarm

types. This result answers the question (3) that we posed in Section VI about user-context's relevance to predict user-preferred application actions. Next, we checked if *Sycophant* could context-enable *Winamp* and thereby evaluate the generalizability of our framework for supporting another widely used desktop application.

C. Short-term study2, Winamp, results

TABLE VI
SHORT-TERM STUDY 2: TEST SET FOR PREDICTING ONE OF WINAMP'S FOUR ACTIONS (PLAY, PAUSE, INCREASE VOLUME, DECREASE VOLUME).

Learning Algorithm Participant ↓	II Zero-R	III J48^{+uc}	IV J48^{-uc}	V XCS^{+uc}	VI XCS^{-uc}	VII XCS better than J48? (V > III)
1	0.4940	0.4940	**0.4337**	0.5185	**0.3951**	TRUE
2	0.5000	0.4630	0.5556	0.7113	**0.3442**	TRUE
3	0.6136	0.5682	0.6136	0.7897	**0.3333**	TRUE
4	0.5306	0.5102	**0.4082**	0.7520	**0.7122**	TRUE
5	0.8061	0.7653	0.8061	1.0000	**0.6559**	TRUE
6	0.7111	0.6889	0.7111	0.8790	**0.5801**	TRUE
7	0.7733	0.7733	0.7733	0.9321	**0.8812**	TRUE
8	0.7315	0.6944	0.7315	0.7660	**0.4926**	TRUE
9	0.6727	0.6727	**0.6545**	0.8519	**0.7593**	TRUE
10	0.5800	0.6200	**0.5800**	0.8741	**0.5667**	TRUE
			3		10	9

Table VI shows the test-set prediction accuracy for one of the four *Winamp*'s actions: play, pause, increase volume by 10 percent, and decrease volume by 10 percent. Column I lists the participants and we show Zero-R's performance in Column II. J48's performance with and without user-context is shown in Columns III and IV, respectively. Similarly we show *XCS*' performance with and without user-context in Columns V and VI.

Comparing the prediction accuracy for various participants in Column II with Column III, we see that the decision-tree's (J48) performance is worse than the base-rate (except for participant-10). Similarly comparing Columns II and V shows that, unlike J48, *XCS* outperforms Zero-R (base-rate). When we compare Columns III and V, we note that *XCS* significantly outperforms J48 for all our 10 study participants. *XCS*' superior predictive accuracy shows that it is a better learner than either Zero-R or the decision-tree learner for predicting participant preferred *Winamp* actions. Similar to our first short-term study, removing user-context degrades J48's performance for four participants (1, 4, 9, 10) thereby emphasizing the importance of external user-context (motion and speech) for learning these participants' preferences.

For participant-1, all the three machine learners had a low predictive accuracy. We examined participant-1's data and found that this participant's variation in *Winamp* action preferences was greater when compared to any other participant in this study. We need to collect additional user-context exemplars from this participant to investigate the low predictive accuracy of all three machine learners. Also, removing user-context degrades *XCS*' ability to accurately predict a user-preferred *Winamp* action. This result again highlights the importance of user-context information for an application to better predict user preferences. *Sycophant* was thus successfully able to context-enable *Winamp* and predicted participant preferences for *Winamp* actions. These results answer the final question (5) that we posed regarding Sycophant's generalizability across multiple applications in Section VI. We next discuss some additional insights which we gained from these three user studies.

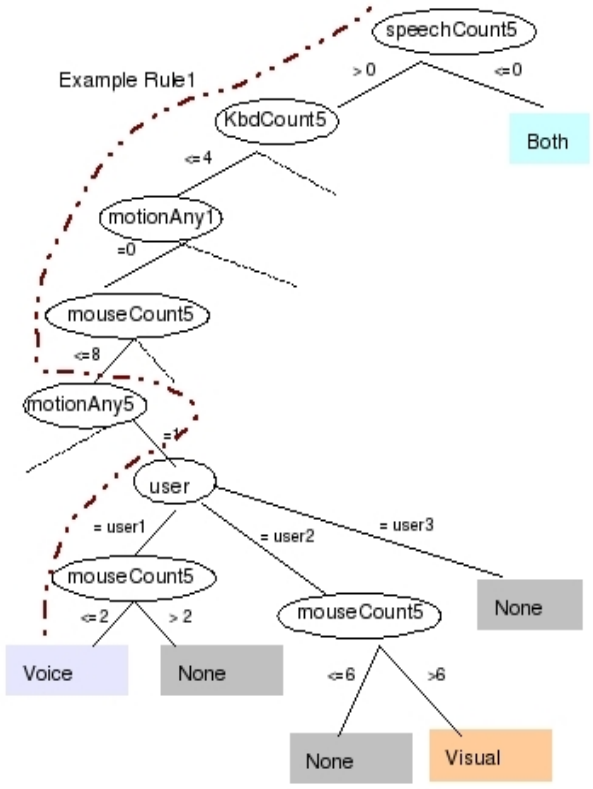

Fig. 7. An Example Rule predicting a Winamp action for participants 1, 2, and 3, respectively

D. Discussion

Our three user studies showed that *XCS* significantly outperformed a widely used decision-tree learner for predicting participant preferred interface actions for a calendar and a media player. To investigate *XCS*'s superior predictive accuracy, we examined the classifiers generated for each interface action for individual participants. We found that *XCS* generated an equal number of classifiers covering each class for a participant. For example, user7 had 148 classifiers (rules) covering every *Winamp* action. For other participants too, we found that each class had roughly an equal number of classifiers covering it. We attribute XCS' superior performance (when compared to a decision tree learner) to its ability of forming complete maps of its payoff landscape, emphasizing and discovering classifiers that accurately generalize over sets of inputs, and a covering mechanism that assists with low-volume data [1]. These mechanisms result in more *XCS* rules covering each class (when compared to a decision-tree learner, J48) and hence explains its higher test-set predictive accuracy.

When we examined the decision trees generated for participants in all our user studies, we found that for most of the participants, if they had preferences, each participant had a *different* set of rules predicting the *same* interface action preference. For example, we found that the rules predicting participant-3's preference for pausing *Winamp*'s were different from the rules predicting participant-4's preference for the same *Winamp* action. This result emphasizes that users have *different* preferences for the *same* interface actions and hence the need to personalize applications to individual users.

We next examined the decision trees to interpret how the action predictions were made for our study participants. Figure 7

shows a part of a decision tree generated for participants in our first short-term user study (*Google Calendar*). To interpret "Example Rule 1", we start at the root node of the tree and traverse the tree along the dotted line towards a leaf node. The rule here checks to see if there was any speech in the last five minutes (speechCount > 0), next examines if keyboard usage was low (Count5 = $4/20$), and monitors for low motion activity (motionAny1 = 0). The rule also checks for low mouse usage (mouseCount5 < 8). and examines the user-id to predict an alarm type. This is an instance of personalization where the alarm type prediction is user-based. From this decision-tree snapshot, we infer that if there was speech, low keyboard and mouse usage, participant-1 preferred a voice alarm if the mouse was not used and no alarm if mouse was used. At the same tree level (user), unlike user-1, user-2 preferred a visual alarm if mouse was used and preferred not to be interrupted (no alarm) with low mouse usage. In the same scenario, user-3 preferred to be not interrupted. The user-context features present at different nodes in the decision-tree emphasize the importance of external user-context (speech derived user-context features) to predict an appropriate alarm for these three users. We have found similar evidence of personalization to individual users where user-context has played a significant role in all our user studies. All study related files are available at *Sycophant's* website [10].

VII. Conclusion and Future Work

In this paper, we first showed that *XCS*, within *Sycophant*, our user modeling framework, could harnesses external user-related information (motion and speech) from their desktop environments to enable widely-used applications such as *Google Calendar* and *Winamp* to learn an individual user's preferences for various interface actions. Second, our three real-world user studies demonstrated that *XCS* significantly outperforms a decision-tree learner on the inteface-action prediction tasks for different participants. In addition to showing that *XCS* is a viable technique to predict interface action preferences for specific users, this result also indicated that we could successfully learn user preferences for multiple participants and applications thereby establishing *Sycophant's* generalizability. Our third result showed that removing user-context significantly degraded both *XCS*' and a decision-tree learner's action prediction accuracy. This performance degradation highlighted the need for applications to access external user-related information to better learn user preferences.

The work discussed in this article suggests further investigations into feature selection and different approaches for soliciting user-feedback. We would like to examine additional user-context features in addition to motion and speech to better predict user preferred actions. Additionally, we also wish to experiment with different approaches that solicit user feedback in a less intrusive manner.

Currently, we are collecting long-term usage data from participants using *Google Calendar* for their daily activities. We also intend to conduct a similar long-term study for *Winamp* in the future. We expect the results from these studies to further substantiate our hypothesis of using an *XCS*-based approach to better predict user-preferences for common desktop applications by harnessing user-related external contextual information. We believe that such user-preference learning can improve the quality of human-computer interaction.

ACKNOWLEDGMENT

The authors thank all the participants in our user studies for their time and patience. We thank three anonymous reviewers for their insightful comments that helped us to improve this article. This work was supported in part by contract number N00014-0301-0104 from the Office of Naval Research and the National Science Foundation under Grant No. 0447416.

REFERENCES

[1] S. W. Wilson, "Classifier fitness based on accuracy," *Evolutionary Computation*, vol. 3, no. 2, pp. 149–175, 1995. [Online]. Available: citeseer.ist.psu.edu/wilson95classifier.html

[2] T. Kovacs, *Bibliography of Real-World Classifier Systems Applications*. Springer, April 2004, vol. 150, pp. 300–305. [Online]. Available: http://www.cs.bris.ac.uk/Publications/Papers/2000096.pdf

[3] A. Shankar, "Simple user-context for better application personalization," Master's thesis, University of Nevada, Reno, 2006.

[4] A. Shankar, S. J. Louis, S. Dascalu, L. J. Hayes, and R. Houmanfar, "User-context for adaptive user interfaces," in *IUI '07: Proceedings of the 12th international conference on Intelligent user interfaces*. New York, NY, USA: ACM, 2007, pp. 321–324.

[5] A. Shankar, "Sycophant: A context based generalized user modeling framework for desktop applications," Ph.D. dissertation, University of Nevada, Reno, Reno, Nevada, Aug. 2008. [Online]. Available: http://www.cse.unr.edu/~syco

[6] Google calendar (accessed apr. 5, 2008). Http://www.google.com/calendar.

[7] Winamp media player (accessed apr. 5, 2008). Http://www.winamp.com.

[8] J. R. Quinlan, *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1992.

[9] Open source initiate (accessed apr. 5, 2008). Http://www.opensource.org/.

[10] Sycophant (accessed apr. 5, 2008. Http://www.cse.unr.edu/~syco.

[11] A. Shankar, J. Quiroz, S. M. Dascalu, S. J. Louis, and M. N. Nicolescu, "Sycophant: An api for research in context-aware user interfaces," in *ICSEA '07: Proceedings of the International Conference on Software Engineering Advances (ICSEA 2007)*. Washington, DC, USA: IEEE Computer Society, 2007, p. 83.

[12] P. J. Brown, J. D. Bovey, and X. Chen, "Context-aware applications: from the laboratory to the marketplace," *IEEE Personal Communications*, vol. 4, no. 5, pp. 58–64, October 1997. [Online]. Available: http://www.cs.kent.ac.uk/pubs/1997/395

[13] P. J. Brown, "The stick-e document: a framework for creating context-aware applications," in *Proceedings of EP'96, Palo Alto*. also published in it EP–odd, January 1996, pp. 259–272. [Online]. Available: http://www.cs.kent.ac.uk/pubs/1996/396

[14] A. K. Dey, G. D. Abowd, M. Pinkerton, and A. Wood, "Cyberdesk: A framework for providing self-integrating ubiquitous software services," in *ACM Symposium on User Interface Software and Technology*, 1997, pp. 75–76. [Online]. Available: citeseer.ist.psu.edu/dey98cyberdesk.html

[15] A. K. Dey, G. D. Abowd, and D. Salber, "A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware appplications." *Human Computer Interaction*, vol. 16, 2001.

[16] J. Fogarty, S. E. Hudson, and J. Lai, "Examining the robustness of sensor-based statistical models of human interruptibility," in *CHI '04: Proceedings of the SIGCHI conference on Human factors in computing systems*. New York, NY, USA: ACM, 2004, pp. 207–214.

[17] I. H. Witten and E. Frank, *Data Mining: Practical machine learning tools with Java implementations*. San Francisco, USA: Morgan Kaufmann, 2000.

[18] P. D. Adamczyk and B. P. Bailey, "If not now, when?: the effects of interruption at different moments within task execution," in *CHI '04: Proceedings of the SIGCHI conference on Human factors in computing systems*. New York, NY, USA: ACM, 2004, pp. 271–278.

[19] S. T. Iqbal and B. P. Bailey, "Understanding and developing models for detecting and differentiating breakpoints during interactive tasks," in *CHI '07: Proceedings of the SIGCHI conference on Human factors in computing systems*. New York, NY, USA: ACM, 2007, pp. 697–706.

[20] S. T. Iqbal and E. Horvitz, "Disruption and recovery of computing tasks: field study, analysis, and directions," in *CHI '07: Proceedings of the SIGCHI conference on Human factors in computing systems*. New York, NY, USA: ACM, 2007, pp. 677–686.

[21] J. Shen, L. Li, T. G. Dietterich, and J. L. Herlocker, "A hybrid learning system for recognizing user tasks from desktop activities and email messages," in *IUI '06: Proceedings of the 11th international conference on Intelligent user interfaces*. New York, NY, USA: ACM, 2006, pp. 86–92.

[22] A. N. Dragunov, T. G. Dietterich, K. Johnsrude, M. McLaughlin, L. Li, and J. L. Herlocker, "Tasktracer: a desktop environment to support multi-tasking knowledge workers," in *IUI '05: Proceedings of the 10th international conference on Intelligent user interfaces*. New York, NY, USA: ACM, 2005, pp. 75–82.

[23] J. Fogarty and S. E. Hudson, "Toolkit support for developing and deploying sensor-based statistical models of human situations," in *CHI '07: Proceedings of the SIGCHI conference on Human factors in computing systems*. New York, NY, USA: ACM, 2007, pp. 135–144.

[24] S. W. Wilson, "Mining oblique data with XCS," *Lecture Notes in Computer Science*, vol. 1996, pp. 158–??, 2001. [Online]. Available: citeseer.ist.psu.edu/article/wilson00mining.html

[25] S. Saxon and A. Barry, "XCS and the monk's problems," in *Proceedings of the Genetic and Evolutionary Computation Conference*, W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, Eds., vol. 1. Orlando, Florida, USA: Morgan Kaufmann, 13-17 1999, p. 809. [Online]. Available: citeseer.ist.psu.edu/article/saxon99xcs.html

[26] J. H. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. Cambridge, MA, USA: MIT Press, 1992.

[27] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading, MA: Addison-Wesley, 1989.

[28] S. W. Wilson, "ZCS: A zeroth level classifier system," *Evolutionary Computation*, vol. 2, no. 1, pp. 1–18, 1994. [Online]. Available: citeseer.ist.psu.edu/wilson94zcs.html

[29] T. Kovacs, *XCS Classifier System Reliably Evolves Accurate, Complete, and Minimal Representations for Boolean Functions.* Springer-Verlag, August 1997, pp. 59–68.

[30] Stewart W. Wilson, *Classifier Systems and the Animat Problem*, ser. Machine Learning. Hingham, MA, USA: CCSDS, 1987, no. 3. [Online]. Available: http://dx.doi.org/10.1023/A:1022655214215

[31] S. W. Wilson, "Generalization in the XCS classifier system," in *Genetic Programming 1998: Proceedings of the Third Annual Conference*, J. R. Koza, W. Banzhaf, K. Chellapilla, K. Deb, M. Dorigo, D. B. Fogel, M. H. Garzon, D. E.Goldberg, H. Iba, and R. Riolo, Eds. University of Wisconsin, Madison, Wisconsin, USA: Morgan Kaufmann, 22-25 1998, pp. 665–674. [Online]. Available: citeseer.ist.psu.edu/wilson98generalization.html

[32] M. V. Butz, "XCSJava 1.0: An Implementation of the XCS classifier system in Java ," Tech. Rep. 2000027, 2000. [Online]. Available: citeseer.ist.psu.edu/butz00xcsjava.html

[33] R. Likert, *A technique for the measurement of attitudes*. New York: New York, 1932.

[34] J. L. Devore, *Probability and Statistics for Engineering and the Sciences*. Duxbury, 1999.

[35] A. Shankar and S. J. Louis, "Better personalization using learning classifier systems," in *Proceedings of the 2005 Indian International Conference on Artificial Intelligence, December 20-22 2005, Poona, INDIA*, 2005, pp. 115–120.

Anil Shankar is a Ph.D. student in the Evolutionary Computing Systems Laboratory. He is working on using evolutionary computing techniques for adaptive user interfaces research. More information on his current work is available at his website http://www.cse.unr.edu/~anilk and he can be reached at anilk@cse.unr.edu.

Sushil J. Louis is a professor and director of the Evolutionary Computing Systems Laboratory, in the Department of Computer Science and Engineering, University of Nevada, Reno, Nevada 89557, USA. Dr. Louis received the Ph.d. from Indiana University, Bloomington, in 1993 and is a member of the IEEE and ACM. More information on his current work is available at his website http://www.cse.unr.edu/~sushil and he can be reached at sushil@cse.unr.edu.