Sycophant: An API for Research in Context-Aware User Interfaces

Anil Shankar, Juan Quiroz, Sergiu M. Dascalu, Sushil J. Louis, Monica N. Nicolescu
Department of Computer Science and Engineering
University of Nevada
Reno, Nevada 89557
{anilk, quiroz, dascalus, sushil, monica}@cse.unr.edu

***Abstract*—Research in context-aware user interfaces aims to improve human-computer interaction by providing more effective, smarter and user-friendlier solutions for computer applications. Currently, software available for performing such research and developing context-aware interfaces is very limited both in scope and possibilities of extension. Sycophant was designed with two objectives in mind: first, to allow easy insertion of new features and capabilities needed for conducting research and, second, to provide a reusable, readily available programming resource for developing new context-aware interactive software applications. Available as open source software, Sycophant's API and the calendaring application we created using it are presented in this paper in terms of functional capabilities, high level architecture, detailed design, and results of use. Procedural steps for developing new context-aware user interfaces using our API are also described in the paper.**

I. Introduction

Present day computer applications rely on the activity of an internal clock, keyboard and mouse to provide input or *context* to interact with a user. Applications that rely on such meager contextual information are only partially aware of a user and her environment. For example, consider the scenario of Jill listening to music on her media player. Jill pauses her media player if the phone rings in her office, and turns the volume down if she is talking with someone in her office. In a similar situation, Jack prefers to turn down the volume on his media player if the phone rings in his office; he pauses the media player while talking with someone in his office. Application action preferences not only vary with the context in which the application is used, but they are also different from user to user. Jill's interaction with her media player could be a lot more effective if her media player harnessed additional contextual information from her environment and adapt its behavior to turn the music volume down whenever she is talking with someone in her office.

There are no currently available Application Programming Interfaces (APIs) to access such contextual information from a user's environment. For example, if one wanted an API to harness speech-related contextual information from Jill's environment and use this information to enable her media player to learn Jill's preferences for turning the volume down or pausing the music, there are no readily available API's for one's use. There is a necessity for a software environment that gives researchers in context-aware user interfaces access to APIs for extracting contextual information from different sensors. A user's environment is a rich source for simple contextual information such as the existence of motion or speech, in addition to the activity of an internal clock, keyboard and mouse. If such APIs were available, then one could use a speech sensor and, for example, detect speech in the last ten minutes or record the quantity of speech activity in the last five minutes. Jill's media player could potentially use speech sensor data to learn her context-based preferences for different situations.

In this paper, we give details about using our *Sycophant API* to extract user-related contextual information from different sensors and develop context-aware user interfaces. To be more precise, we use the name Sycophant for both the API we have developed and the context-aware software environment (centered on a calendaring application) that we have built using the API. However, because the API is more generic in scope and provides the framework on which other, new context-aware user interfaces can be built, this paper is focused primarily on the Sycophant API rather than on Sycophant calendar-centered software environment.

Sycophant API allows developers to create different user-related features and employs these features to build a user model for individual users. We can harness this user model to learn preferences for different applications. For the work presented here, we provide results from using our API in a calendaring application to learn alarm preferences for different users. Sycophant API allows access to simple sensor information from a user's environment such as motion, speech, keyboard activity, and mouse usage. We extract different contextual features from these sensors to check, for example, if there was any activity in the last five minutes and, if there was, the amount of sensor activity in this period of time. Sycophant API is reused to extract similar features for the last minute before which our calendar generates an alarm. Related work by Shankar et. al [1], [2] and Fogarty et. al [3], [4] gives more details about the construction of such features for context aware interfaces. In this paper we focus on highlighting the utility of Sycophant API for such tasks.

Our goal is to provide researchers in context-aware user interfaces access to APIs such that they can specify sensors to be used and features to be extracted, and use the information collected to enhance the adaptive behavior of

different applications. We believe that such context-enabled applications capable of learning user preferences have a very high potential for improving the quality of human-computer interaction (HCI).

To summarize, the main contributions of this paper are the following: first, it provides an account of a unique, operational context-aware calendar-centered software environment used both for research and actual office work and, second, offers details on an API that can be readily used for developing new context-aware applications (for example, a media player or an email browser able to learn user preferences and behave to their satisfaction).

This rest of this paper is organized as follows. Section II provides background on currently available open source APIs and their applicability for research in context-aware user interfaces. We also talk about leveraging the Sycophant API to enable Google Calendar to adaptively generate alarms for different users [5] . Section III describes the Sycophant context-aware learning environment and the components of its layered architecture. Details of functionality and user interface design are also provided in this section. Class diagrams presenting the organization of our API and procedural steps for using the API in developing a particular software application are given in Section IV. Results from using the Sycophant API and environment in our research are given in Section V. Finally, in Section VI we present our conclusions and outline directions of future work.

II. Background and Related Work

We are not aware of any readily available APIs that allow researchers in context aware user interfaces to extract context features from different sensors. Our survey revealed two APIs/software packages that come the closest to our Sycophant API. The first is a software package provided by Carolina Computer Assistive Technology group at the University of North Carolina-Chapel Hill [6]. Their approach focuses mostly on the development of applications for people with disabilities. The *pyHook* library included in the packages wraps low-level mouse and keyboard events in MicroSoft's Windows Hooking API. This API cannot be used on Linux platforms and there is no facility to directly extract sensor features. The second, Fogarty's *Subtle*, is a software package that collects data from a note-book computer's closing, opening, mouse-click, audio analyses, and WiFi sensing activities [3]. Subtle can create new features and operators based on a context feature's type and history of usage. However, Subtle is currently not available for use on Linux platforms. Our Sycophant-API is transparent to the operating system and is currently usable on both Windows and Linux [7].

The next section gives a brief description of Sycophant's architecture. We used Sycophant in a study involving three users for a period of four to six weeks. Sycophant enabled a calendaring application to adaptively generate alarms for these three users [1]. Results of this study showed that our API successfully extracted user-related contextual features to enable the calendaring application to adaptively generate alarms for these three users. In a second short-term study involving ten users, Sycophant was again successful in adaptively generating hints (reminders) to study subjects participating in four article-reading sessions [2]. The results of these studies have demonstrated Sycophant utility for research in context-aware user interfaces and environments. Encouraged by these results, we are currently investigating generalizing Sycophant by using it for Google Calendar (a web-based calendar) and XMMS (media player on Linux) [8]. We intend to deploy the user-context software on at least ten users' PCs and collect long term data from these users for investigating the usage of contextual information from a user's environment to help desktop applications personalize their behavior to individual users.

III. Overview of Sycophant Software Environment

We have used the Sycophant API to build the Sycophant context-aware interactive software environment. The target application for which we wanted to learn user preferences and adapt application behavior to these preferences has been a calendar. Nevertheless, the principles of building a context-aware environment such as Sycophant are largely independent of the target application.

Thus, in this section we first describe use cases that specify functionality provided by the Sycophant environment. Next, we detail the components (sub-APIs) of its layered software architecture. We finally conclude the overview of the environment with details of the interface designed to capture user feedback.

A. Functionality

The major actors that interact with a context-aware learning environment such as Sycophant are the following: the user of the environment (an environment that embeds a target user application such as a calendaring program or a media player), the sensors used to collect data relevant to user behavior (sensors such as motion, keyboard, mouse, and speech sensors), and the time, which basically provides timestamps needed in analyzing research data.

In Sycophant's case these actors, together with the use cases in which they are involved, are shown in Figure 1. Actors and use cases are elements of the Unified Modeling Language (UML) that capture system behavior as seen from outside the system [9] and [10]. They are powerful model elements that can cover the entire system behavior in a similar way pieces of a puzzle, when all put correctly together, make clear and complete the image hidden in the puzzle.

Due to space limitations, only the main functionality of the Sycophant environment is shown in Figure 1. Nevertheless, it serves us to outline from an operational perspective the organization of a context-aware interactive software environment. Note that all use cases are triggered by actors and, at this level of representation, the system can be seen as a black box (i.e., the way use cases are implemented in the system is irrelevant to the actors) [11]. Note also that in Figure 1 all

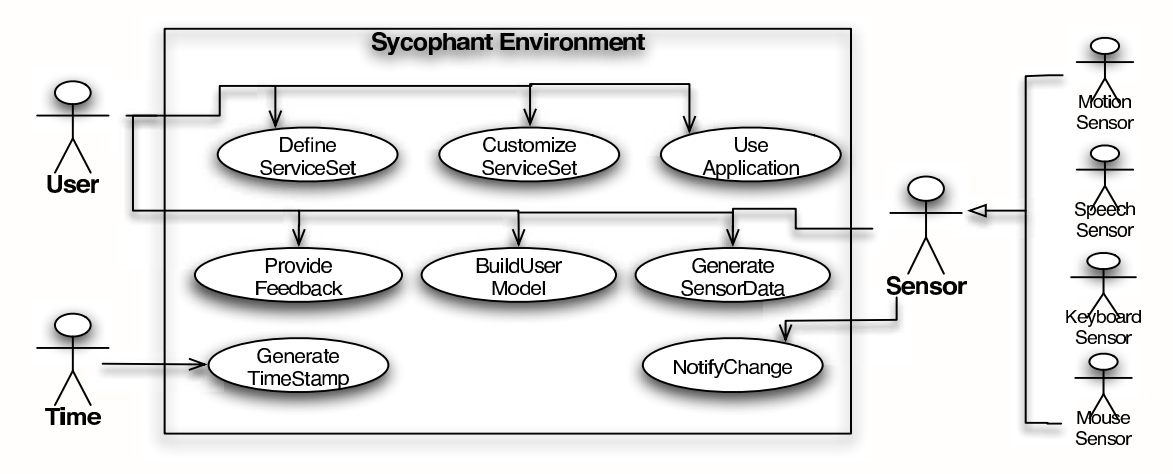

Fig. 1. Use case diagram of the Sycophant user-centered context-aware learning environment

sensor actors inherit from an abstract actor, denoted *Sensor*. The *GenerateTimeStamp* use case indicates that the *Time* actor interacts with the system by associating timestamps with data items collected from sensors. The sensors themselves interact with the system by notifying changes, e.g., when a web-camera detects motion in the vicinity of the computer (this is captured in the *NotifyChange* use case). When requested (by the *User*) the sensors also provide sensor data, fact indicated by their involvement in the *GetSensorData* use case. The *User* of the system initiates most use cases. These are as follows:

- *DefineServiceSet* allows the user to specify the types and the number of sensors available in the system;
- *CustomizeServiceSet* is invoked for selecting a subset of available sensors to be used in an actual operation of the system (e.g., in a research experiment, or over of a specified period of time of using the application);
- *GetSensorData* allows the user to specify the parameters of data collection, including sampling intervals;
- *BuildUserModel* provides the creation of the user model (i.e., a model of user behavior) based on data collected;
- *UseApplication* is the actual use of the target application embedded in the context-aware environment (e.g., of the calendar, with all its alarm types that take into consideration user preferences);
- *ProvideFeedback* solicits feedback from the user during the use of the application on various aspects of use that help the system learn user preferences (more details are given in subsection III-C, which presents the user interface used for this purpose).

B. Architecture

Figure 2 shows the architecture of our Sycophant user-context aware learning environment, built using the Sycophant API. Sycophant API consists of four components (sub-APIs): Sensors API, Context API, Learning Services API, and Application-Level API. Sensors API interfaces different sensors (motion, speech, keyboard, mouse) with a user's environment. For example, we can interface with a web-camera (motion sensor) and create a motion detection service at the sensor's level. We can similarly interface with a microphone, a keyboard, or a mouse. Different sensors store their data in the

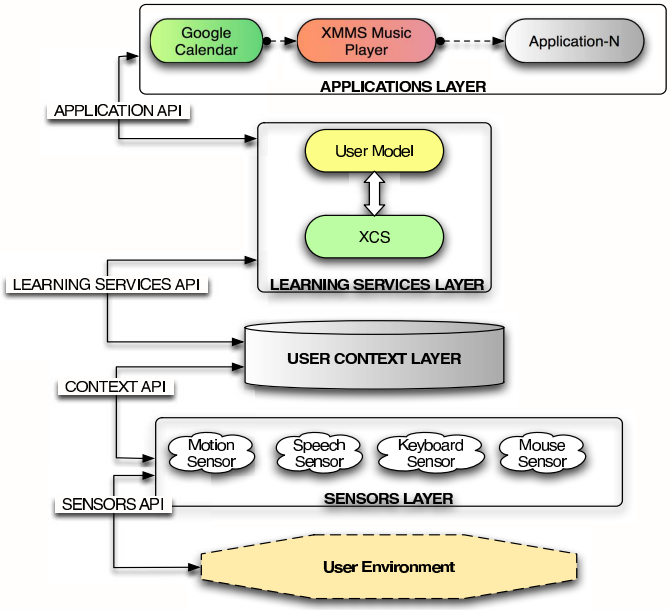

Fig. 2. Four-layer architecture of the Sycophant software environment

User-Context layer. We can extract user-context features from this data using the Context-API. For example, if we want to count the number of times the motion detector was active in the last 10 minutes we can use the Context-API to extract a Count-10 feature that accomplishes this task. Using the User Context API, we can similarly extract different sets of features from all the sensors. Section IV provides more details on how to create a user-context data set extracting different features from various sensors.

We can use the Learning Services API to select a machine learning algorithm for generating an application-specific user

model based on user-context data collected in the User Context layer. A user model maps user-related contextual features to applications. We can use this API to select a decision-tree learning algorithm for learning Jill's music volume preferences based on speech activity detected in her environment. Learning Services API allows use to plug-in any other machine-learning algorithm for generating user models that reflect their application action preferences.

The Application API provides access to the user-model (generated at the Learning Services layer) for predicting a user preferred application action. We can use this API to enable Jill's media player to predict when the volume should be turned down based on Jill's model built using different sensor data. Thus, the Sycophant API plays a key role in aiding applications to adapt their behavior to individual users.

C. Interface for User Feedback

In the Sycophant environment a lot of program execution occurs "behind the scenes" and typically only the target ("wrapped") application (i.e., a calendar software utility) is accessed externally. However, to learn user preferences, Sycophant asks the user to provide feedback during her use of the application. Details of the interface used in our calendaring version of the Sycophant environment are shown in Figure 3. Note that the quote , similar to the *fortune* program on Linux, is included as an added motivation for the user to provide feedback to the learning system.

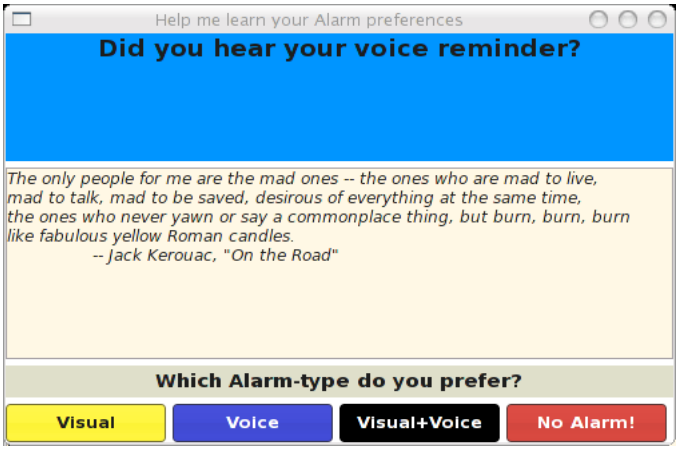

Fig. 3. Interface for user feedback shown after a voice alarm has been issued

IV. Sychophant API: Component Details and Application Set-up Procedure

Due to space limitation not all details pertaining to the Sycophant API are presented in this paper. For complete information, the reader is invited to download the Sycophant software publicly available via [7]. Nevertheless, in this section we provide a thorough coverage of the Sycophant API components Sensor API and Application API. These are described using the class diagrams created to define them. In addition, we illustrate the use of the API on setting up our calendaring application. The steps followed in doing this are general and can be used for implementing other context-aware applications.

A. Sensors API

Figure 4 shows the class diagram of the Sensor API component of the Sycophant API. The *SycoMonitor* class contains and manages multiple instances of user-context sensors. SycoMonitor has attributes that reflect the status of different user-context sensors. For example, *motionActive* checks if the motion detection sensor is active. SycoMonitor uses similar status flags for keyboard, mouse and speech sensors. The attribute *runInterval* specifies how often the context sensors are polled for raw data. In SycoMonitor, the *createPeripherals* method initializes the keyboard and mouse peripherals; *createMotionSensor* and *createSpeechSensor* initialize motion and speech sensors, respectively. All these three methods create instances of the *UserContextSensor* class. The UserContextSensor has attributes: *logFile* to log a sensor's data and *logInterval* to specify how often the sensor data needs to be logged. The attribute *startDetection* is used to start detecting activity from a sensor and the attribute *stopDetection* to stop a sensor's activity detection. The *runThread* method starts a thread that continuously tracks sensor activity. Three sensor specific classes are derived from UserContextSensor. The *PeripheralsActivityDetector* class manages keyboard and mouse sensors. Next, the *MotionDetector* class manages motion detection. It has attributes to store the previous image (*previousImage*), the current image (*currentImage*), the minimum allowed threshold for the difference between the two images, and the program to use for grabbing images from a web-camera (*imageGrabber*). Lastly, the *SpeechDetector* class manages speech activity detection – its *speechSoftwareCmd* specifies the speech recognition software to be used for detecting speech from a user's environment.

B. Application-Level API

Figure 5 shows the class diagram of the Application API component of the Sycophant API. The *GCalMonitor* class manages alarms from a user's calendar. The method *checkForAlarms* checks for any current or pending appointments every *alarmCheckInterval* seconds. The *GCalProcessor* gathers the calendaring data from a user's calendar file (*getCalendarData* method), accesses a user-preferred alarm type (*getPredictedAlarm()*) and generates an alarm for a current or pending appointment using the *generateAlarms()* method. *GCalParser* parses calendar data from a user's calendar file. The methods *checkIfDaily, checkIfWeekly, checkIfMonthly* check for daily, weekly, and monthly repeating appointments, respectively. The *AlarmGenerator* class generates different types of alarms and notifications. The *generateFortuneCookie* generates a fortune cookie (an interesting quote) along with the alarm generated for an appointment using the attribute *fortuneCookieText*. User context information from sensors (sensor features related to motion, speech, keyboard and mouse) sensors get logged using the *writeCurrentUserContext* method. The method *generateAlarm* generates an alarm for a user with the attribute

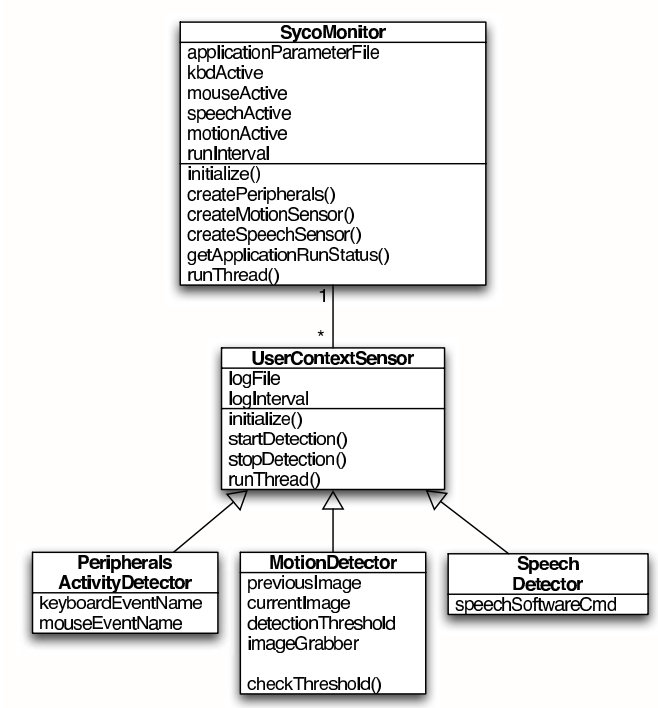

Fig. 4. Class diagram of Sycophant's Sensors API

.

alarmText. A user feedback for her preferred alarm type is stored in the attribute *userFeedbackRequest* using the method *getUserFeedback*. This class also has methods related to alarm prediction: *getPredictedAlarmType* uses the *predictedAlarmType* to record an alarm type predicted for a user by executing a machine learning algorithm trained on that user's context data, and *writePredictedAndActualAlarms* logs to a file the predicted alarm and the user-preferred alarm type obtained from the user's feedback.

The classes *VoiceAlarm* and *Visual Alarm* inherit from the *AlarmGenerator* class and generate voice and visual alarms, respectively. For example, Figure 3 shown previously presents the interface of the feedback form displayed after a voice alarm was issued. A text to speech generator voices out appointment text as a voice alarm for a user. The *AlarmPredictor* class obtains the current sensor values using the attribute *currSensorValues* and checks for existing user context data using *checkForUserContextData* information. Based on the availability of this context information, this class helps generate a user model that reflects her preferences for alarm types using the *buildUserModel* method. The method *predictAlarm* accesses a machine-learning algorithm, feeds it with the current sensor values and obtains a predicted alarm type using the generated user model.

Most of target application-specific parameters (in this case, Google Calendar related parameters) are set in an *AppParameters* class (in our case, in the *GCalParameters* class) The following are the different attributes: *wgetCmd* obtains a user's calendar (*userName*) using the attribute *googleCalendarLink* which specifies the user's calendar web link. Alarms are checked every *alarmCheckInterval* seconds and generated *preAlarmTime* seconds before the actual time of the appointment. Visual alarms and feedback request pop-ups auto close after *autoCloseAlarmInterval*. The *webcamImageGrabberApp* attribute specifies the application to use for grabbing images from the user's web-camera and the *textToSpeechGeneratorApp* attribute specifies the text to speech generator used for generating voice alarms. The *gCalUserModel* specifies the location of a user model that reflects the user's alarm type preferences. This model is generated using the machine learning algorithms located in the *machineLearnersRootDir*. The *machineLearnersPath* specifies the machine learning algorithm to use for generating a user model. This algorithm gets trained on *userContextTrainingSet*, which is the training data and an alarm is predicted for the current set of context values stored in the *userContextTestSet*. Methods *setGCalParameters* and *setLearningParameters* set the parameters related to the calendar and the learning algorithms for a user, respectively.

Figure 5 shows classes related to extracting user context features from the raw sensor data. The class *UserContextCreator* has methods to set which sensors to use (*setSensors*), indicate the features to extract (*setFeatures*), and extract context data using these features from the raw sensor data (*getUserContextData()*). The *FeatureExtractorClass* extracts the context features used by *UserContextCreator* class. It has methods to calculate how many times the a sensor was active, if it was active during any or all of those minutes. The *FeatureExtractor* uses *SensorTail* class which mimics the *tail* command on a Linux/Unix operating system and obtains the specified number of lines (*nLinesToGrab*) from a sensor's log file *sensorLogFile*. The data in this log file is a time-stamp indicating the activity of a sensor.

C. Sycophant API in use: Example of application set-up procedure

While next we show how to use Sycophant API for making the Google Calendar "context-aware" it is worth noting that the steps of setting up a sensor like the one described below are the same regardless of the type of target application or the type of sensor involved. These steps can be summarized as follows:

1) Sensor setup
2) Feature-extractor setup
3) Feature set extraction
4) Target application setup

Specifically, code excerpts provided below show how to set up a motion sensor. We can similarly set up the peripherals (keyboard and mouse) and the speech sensor using the following steps.

Listing 1 shows how to set up a sensor and activate it to its log raw data (timestamp value) to a file. We first create a sensor by specifying its name and its associated log file (line 1), then we activate the sensor by calling the start method on it (line 2).

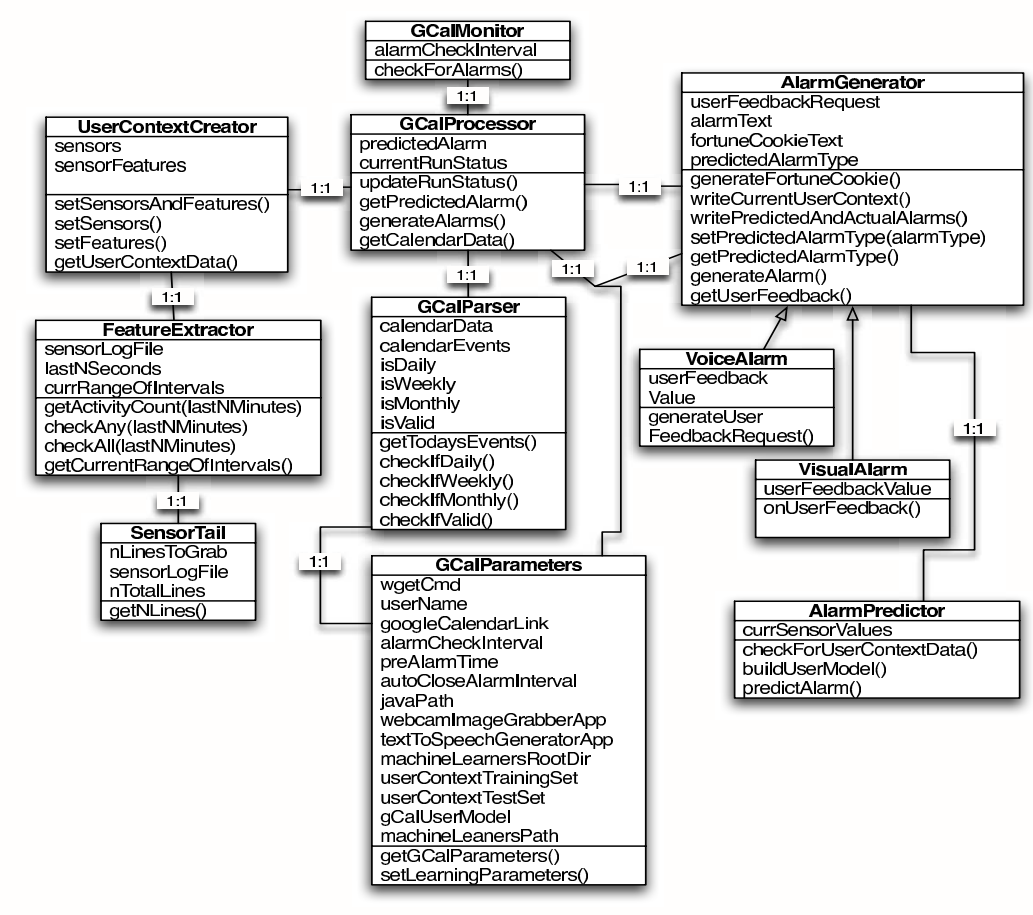

Fig. 5. Class diagram of Sycophant's Application-Level API

Listing 1. Sensor Setup

```
1 motionSensor = Sensor('motion', motionLogFile)
2 motionSensor.start()
```

Listing 2 shows how to extract user-context features from a sensor by setting up a feature extractor. We specify the length of a sensors activity history in the past that we want to examine (line 1) and indicate the type of feature extractor used (line 2). In this case, we want to look at the motion sensors activity in the last five minutes. We can check if the sensor was active during any of the five minutes or in all the five minutes by specifying the *checkAny* and *checkAll* features (lines 3 and 4). We can also check how many times the motion sensor was active in the period of five minutes by specifying the *getCountAll* feature for the motion feature extractor (line 5).

Listing 2. Feature-Extractor Setup

```
1 lastNMinutes = 5 # duration of history check
      for a sensor
2 motionFeatureExtractor = FeatureExtractor(
      motionLogFile)
3 checkAnyNMinutes = motionFeatureExtractor.
      checkAny(lastNMinutes)
4 checkAllNMinutes = motionFeatureExtractor.
      checkAlllastNMinutes)
5 getCountAllNMinutes = motionFeatureExtractor.
      getCount(lastNMinutes)
```

Listing 3 shows how to create user context data using different features extracted from different sensors. More details about the meaning of these features are given in [1].

Listing 3. Extracting Features

```
1 motionFeatures = checkAny-1, checkAll-1,
      checkAny-5, checkAll-5, getCount-5
2 speechFeatures = checkAny-1, checkAll-1,
      checkAny-5, checkAll-5, getCount-5
3 motionUserContextData = UserContextExtractor(
      motionFeatures)
4 speechUserContextData = UserContextExtractor(
      speechFeatures)
```

Listing 4 shows how we use the context sensors for a target application such as the Google Calendar by associating a calendar file (line 1). Figure 5 provides additional details about the target application wrapped within the Sycophant API (specifically, in its Application-Level API component).

Listing 4. Application-Specific Use

```
1 calendarMonitor = GCalMonitor(calendarFile)
2 calendarMonitor.generateAlarms()
```

V. Results

We first deployed Sycophant API prototype to three users in a long term study lasting four to six weeks [1], [12]. A simple calendaring application we authored used Sycophant API to generate four types or alarms for a user. The first alarm type was a visual alarm that displayed the appointment text, the second alarm type was a voice alarm where a text to speech generation system voiced out an alarm for a user, the third alarm type combined both visual and voice alarms, and the fourth alarm type was a no-alarm (the user was not interrupted by any alarm). We tested our hypothesis of using contextual information from a users environment to learn her preferences for alarm types by predicting the alarm type. In our research, the two-class alarm prediction problem is deciding whether or not to interrupt a user with an alarm, and the four-class alarm problem is picking an alarm type to use from the class of four different alarm types. In our initial long term study, Sycophant API enabled us to achieve a prediction accuracy of 87 percent on the two-class problem and 82 percent on the four-class problem using XCS, a learning classifier system [13]. We give details of the learning algorithms used, the experimental methodology, and analysis results in [1].

To check the generalization of our approach, we conducted a short term study with ten users. During the study, the user read an article for the first 30 minutes on our experimental set-up and answered questions related to the article during the last 15 minutes. Our goal again was to predict the alarm type to use for individual users based on user-context data collected from them during the study. We achieved an accuracy of 86 percent on the two-class problem and 88 percent on the four-class problem using XCS. Currently, we have plugged in the Google Calendar [5] into our environment and are deploying it to more users to gather long term usage data. More details about our study, its design and the analysis of the performance of the best and the worst machine learning algorithms on the alarm prediction tasks are presented in [2].

VI. Conclusions and Future work

In this paper, we pointed out to the lack of context-awareness in many current computer applications and explored the possibility of improving human-computer interaction by harnessing contextual information from a users environment. Also, we identified the lack of a platform-independent API for extracting context features for researchers in adaptive user interfaces. Our proposed Sycophant API is designed to remedy this problem since it is available for use on both Windows and Linux. Further, we described the functionality and the 4-layer architecture of our context-aware environment and showed how Sycophant API provides a reusable, open-source resource for personalizing user interfaces. Our Sycophant component APIs and an example of context enabling a target application (Google Calendar) showed the general procedure for accessing, processing, and using context information from a user environment. Our results from previously conducted studies clearly demonstrated Sycophant's utility for research in context aware interfaces. Notably, while at this time Sycophant is primarily a research tool, its transition to a full-fledged end user tool should be straightforward as very little changes are needed in its software.

Clearly, there is a lot of scope for improving the design of our API. We are currently refining the Sycophant API and plan to collect long term sensor data from at least ten more users. Our goal is to test the generalization of our context learning approach on one or more open source applications such as the XMMS media player for Linux. We plan to incorporate this and other new applications within our Sycophant context-learning environment to create adaptive user interfaces that can interact in more effective and more intelligent ways with the users and thus improve their human-computer interaction experience.

Acknowledgements

We thank the ten users involved in our study for their time. 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] A. Shankar, "Simple user-context for better application personalization," *Master's Thesis, University of Nevada, Reno, NV*, 2006.

[2] A. Shankar, S. J. Louis, S. Dascalu, R. Houmanfar, and L. J. Hayes, "User-context for adaptive user interfaces conference," in *Proceedings of the Intelligent User Interfaces Conference, Honolulu, Hawaii, USA*, 2007, pp. 321–325.

[3] J. A. Fogarty, "Constructing and evaluating sensor-based statistical models of human interruptibility," Ph.D. dissertation, Carnegie Mellon University, Pittsburgh, PA, USA, 2006.

[4] J. Fogarty, S. E. Hudson, and J. Lai, "Examining the robustness of sensor-based statistical models of human interruptibility," *Proceedings of the Conference on Human factors in Computing Systems*, pp. 207–214, 2004.

[5] Google Calendar, April 10, 2007, http://calendar.google.com.

[6] Carolina Computer Assistive Technology Group, UNC Assistive Technology, April 10, 2007, http://sourceforge.net/project/showfiles.php.

[7] Sycophant Website, April 10, 2007, http://www.cse.unr.edu/ syco/.

[8] X Multimedia System, April 10, 2007, http://www.xmms.org.

[9] Object Management Group, Unified Modeling Langauge, April 10, 2007, http://www.uml.org.

[10] J. Rumbaugh, I. Jacobson, and G. Booch, *Unified Modeling Language Reference Manual, 2nd Edition*. Pearson Higher Education, 2004.

[11] J. Arlow and I. Neustadt, *UML and the Unified Process: Practical Object-Oriented Analysis and Design*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002.

[12] A. Shankar and S. J. Louis, "Learning classifier systems for user context learning," in *Proceedings of the IEEE Congress on Evolutionary Computation,September 2-5 2005, Edinburgh, UK*, 2005.

[13] A. K. Shankar and S. J. Louis, "Better personalization using learning classifier systems," in *Proceedings of the Indian International Conference on Artificial Intelligence, December 20-22, Poona, India*, 2005.