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.

1. 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 adapted its behavior to turn the music volume down whenever she is talking with someone in her office.

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. In this paper, we outline an Application Programming Interface (API), *Sycophant API*, to access such contextual information from a user's environment. 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. Sycophant API allows developers to create different user-related features and employs these features to build a user model for individual users. An application developer can leverage this user model to learn preferences for different applications. For example, one can use Sycophant 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. Jill's media player could potentially use speech sensor data to learn her context-based preferences for different situations. We provide results from using our API in a calendaring application to learn alarm preferences for different users based on data from different context sensors. Related work by Shankar et. al [9, 12] and Fogarty et. al [7, 6] gives more details about user-context feature-construction 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 our two main contributions: first, we provide an account of a unique, operational context-aware

International Conference on Software Engineering Advances(ICSEA 2007)
0-7695-2937-2/07 $25.00 © 2007 IEEE

calendar-centered software environment used both for research and actual office work. Second, we offer API details 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). The rest of this paper is organized as follows. Section 2 provides background on currently available open source APIs and their applicability for research in context-aware user interfaces. Section 3 describes the Sycophant context-aware learning environment, its four-layer architecture, and functionality. Class diagrams presenting our API organization and procedural steps for using the API in developing a particular software application are given in Section 4. Results from using the Sycophant API and environment in our research are given in Section 5. Finally, in Section 6 we present our conclusions and outline directions of future work.

2 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 [1]. 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 [7]. 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 [4].

The next section briefly describes Sycophant's architecture. We used Sycophant in a study involving three users for a period of four to six weeks and in a second short term study involving ten users. In both these user studies, Sycophant enabled a calendaring application to adaptively generate alarms for the users [9], [12]. The results of these studies have demonstrated Sycophant's 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 (a Linux media player).

3 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. In this section, we describe use cases that specify functionality provided by the Sycophant environment and the components (sub-APIs) of its software architecture.

3.1 Functionality

The major actors interacting with Sycophant are the following: the user of the environment embedding a target user application (calendaring program or a media player), the sensors used to collect data relevant to user behavior (motion, keyboard, mouse, and speech sensors), and the time, which provides timestamps for 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 [3] and [8].

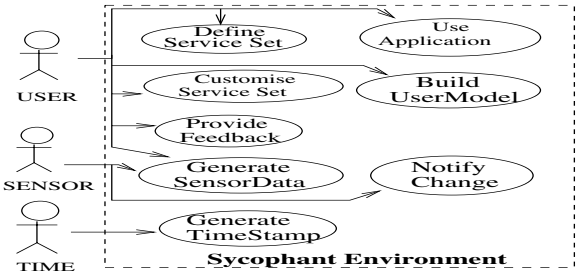

Figure 1. Use case diagram of the Sycophant user-centered context-aware learning environment

Due to space limitations, only the main functionality of the Sycophant environment is shown in Figure 1. Actors trigger all use cases. The system can be seen as a black box since the system use case implementation is irrelevant to the actors [5]. In Figure 1, all sensor actors inherit from the abstract (*Sensor*) actor. The *GenerateTimeStamp* use case shows 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 (captured in the *NotifyChange*

use case). When requested by the *User* the sensors also provide sensor data, indicated by their involvement in the *GetSensorData* use case. The *User* of the system initiates the following use cases: *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 based on context 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 user-preferred alarm types); *Provide-Feedback* solicits feedback from the user during the use of the application on various aspects of use that help the system learn user preferences.

3.2 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 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 4 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.

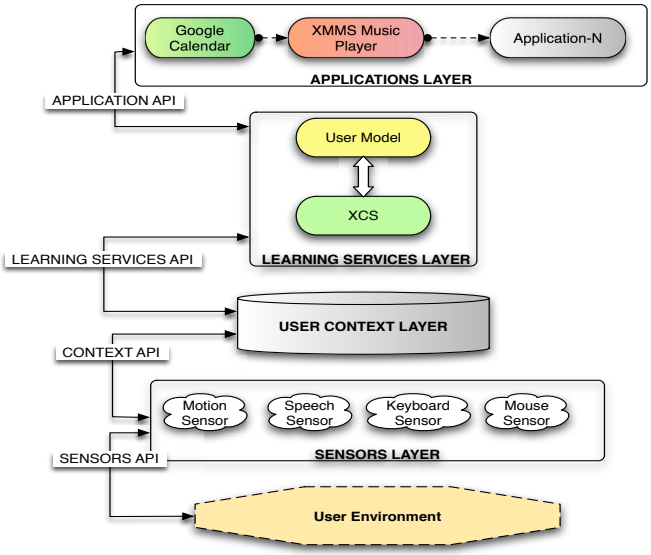

Figure 2. Four-layer architecture of the Sycophant software environment

4 Sychophant API: Component Details and Application Set-up Procedure

Due to space limitation not all Sycophant API details are presented in this paper. For more information, the reader is invited to download the Sycophant software publicly available via [4]. Nevertheless, in this section we use class diagrams to cover Sensor and Application components of Sycophant API. We also illustrate the API use for setting up our calendaring application. The steps followed in doing this are general and can be used for implementing other context-aware applications.

4.1 Sensors API

Figure 3 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 webcamera (*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.

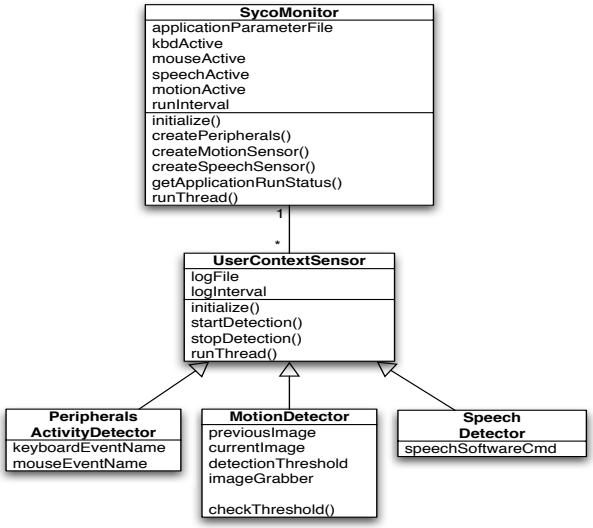

Figure 3. Class diagram of Sycophant's Sensors API

.

4.2 Application-Level API

Figure 4 shows the class diagram of the Application API component of the Sycophant API. The *GCalMonitor* class manages alarms from a user's calendar. This class has methods to check for any current or pending appointments. The *GCalProcessor* has methods for gathering calendaring data from a user's calendar, accessing a user-preferred alarm type and generating an alarm for a current or pending appointment. *GCalParser* parses calendar data from a user's calendar file. The *AlarmGenerator* class generates different types of alarms and notifications. It has methods to to log user context information from different sensors, get user feedback, and predict alarm types.

The classes *VoiceAlarm* and *Visual Alarm* inherit from the *AlarmGenerator* class and generate voice/visual alarms. The *AlarmPredictor* class has methods to get relevant context data, build a user model based on this data that reflects a user's preferences for alarm type, and predict an alarm type leveraging this 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 source code we provide on our website [4] gives more details about each of these application-specific parameters.

Figure 4 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 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.

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

We show how to use Sycophant API to *context-enable* Google Calendar. The setup procedure involves sensor setup, feature-extractor setup, feature-set extraction, and target application setup. This four step procedure is the same regardless of the type of target application or the type of sensor involved. Our 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).

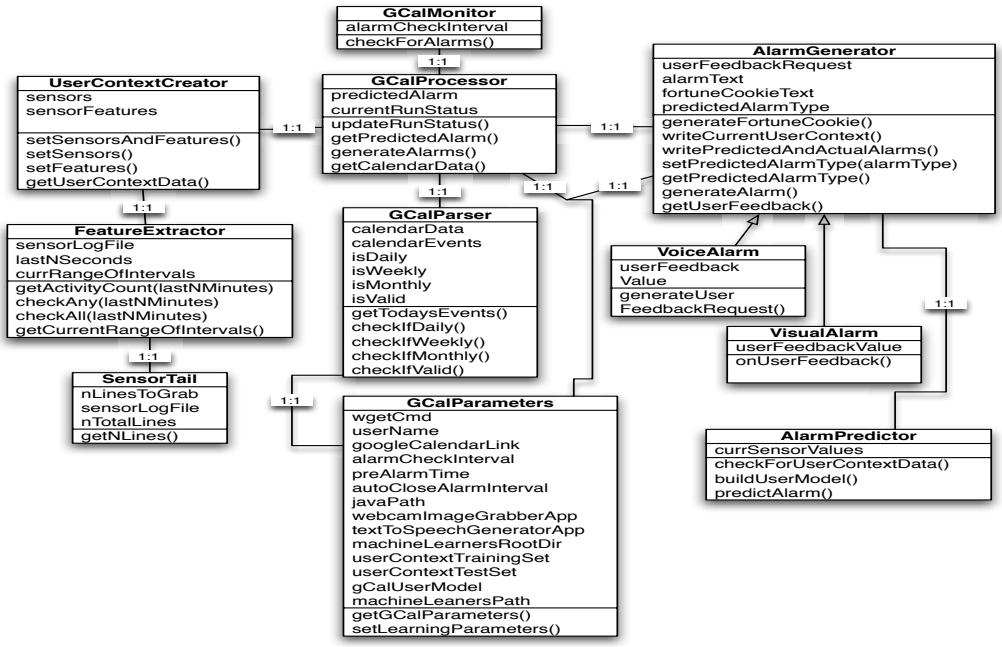

Figure 4. Class diagram of Sycophant's Application-Level API

Listing 1. Sensor Setup

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

Listing 2 illustrates setting up a feature extractor to extract user-context features from a sensor. 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 check the motion sensor's 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 [9].

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 4 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()
```

5 Results

We deployed Sycophant API prototype to three users in a long term study lasting four to six weeks [9, 11]. 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 user's 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 [10]. We give details of the learning algorithms used, the experimental methodology, and analysis results in [9].

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 [2] 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 [12].

6 Conclusions and Future Work

In this paper, we highlighted 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.

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

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

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

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

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

[6] 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*, pages 207–214, 2004.

[7] J. A. Fogarty. *Constructing and Evaluating Sensor-Based Statistical Models of Human Interruptibility*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA, 2006.

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

[9] A. Shankar. Simple user-context for better application personalization. *Master's Thesis, University of Nevada, Reno, NV*, 2006.

[10] A. 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.

[11] 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.

[12] 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*, pages 321–325, 2007.