Context-Aware Experience Sampling for the Design and Study of Ubiquitous Technologies

by

John C. Rondoni

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2003

Author . . . . . .
Department of Electrical Engineering and Computer Science
May 27, 2003

Certified by . . . . . .
Stephen Intille
Changing Places / House_n Technology Director
MIT Department of Architecture
Thesis Supervisor

Accepted by . . . . . .
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

Context-Aware Experience Sampling for the Design and Study of Ubiquitous Technologies

by

John C. Rondoni

Submitted to the Department of Electrical Engineering and Computer Science
on May 27, 2003, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

As computer systems become ubiquitously embedded in our environment, computer applications must be increasingly aware of user context. In order for these systems to interact with users in a meaningful and unobtrusive way, such as delivering important reminders at an appropriate time, their interfaces must be contextually-aware.

This vision of future computer systems and the insight that *the implementation of contextually-aware systems requires contextually-aware analysis and development tools* has motivated the two primary contributions of this work. First, a Context-Aware Experience Sampling Tool has been designed, implemented, and tested. Second, this tool has been used to develop an algorithm that can detect transitions between human activities in office-like environments from planar accelerometer and heart rate data.

The Context-Aware Experience Sampling Tool (CAES) is a program for Microsoft Pocket PC devices capable of gathering qualitative data, in the form of an electronic questionnaire, and quantitative data, in the form of sensor readings, from subjects. This system enables contextually-aware user interactions via real-time modification of the electronic questionnaire based on sensor readings. CAES is publicly available to researchers and is actively being used and evaluated in several studies at MIT.

The algorithm capable of detecting transitions between human activities was evaluated on a data set collected from nineteen subjects with CAES and successfully differentiated continuous activities from activity transitions with 93.5% accuracy. This detector could be used to improve CAES and to develop applications capable of proactively presenting users with information at appropriate times.

Thesis Supervisor: Stephen Intille
Title: Changing Places / House_n Technology Director
MIT Department of Architecture

Acknowledgments

I would like to thank the subjects who invited these devices and sensors into their daily lives. Without their time and patience, this work would not have been possible.

To my colleagues at House_*n*: I hope I have contributed a fraction of what I will take away from my time with you. You have given me a new way to think about the integration of technology, health, and the home. Interacting with such a diverse group has allowed me to understand these challenges more holistically than I could have otherwise.

Stephen:

I am grateful for the direction and support you have given me over the last year. Working with you has enabled me understand how technology can be used to address difficult social problems. I believe it was your knowledgeable advising that led me to make a real contribution with this work.

Isabel:

The success of CAES is a testament to the great work you did in helping us to design the user interface. Without your dedication and hard work CAES would not be nearly as beautiful or usable.

Emmanuel:

Thank you for your patience and help at all hours. I very much appreciate it.

Sachin:

Without your hard work the heart rate monitor would not have been working as well or as soon. Thank you.

Tom (Mr. B):

I will always credit you with teaching me to think critically and argue a point of view. You taught me the power of ideas.

Andrea:

Your help was invaluable over the course of this study and in the writing of this document. I do not know what I would have done without your writing expertise or without such a willing and patient subject.

My Friends:

From MN, BSM, MIT, $\Delta T\Delta$, and elsewhere: you have all supported me through the rough times and been there at the best times. For that I am truly grateful.

Peter:

I am lucky to have you as my brother. I have always been impressed with your capacity for original thought as well as your ability decide what you want out of life and chase it down. I know you will be a success at whatever you decide to do—even if you do not want to be.

Anne:

You are the best sister a guy could hope for. You are an incredibly talented person and still have the dedication to work hard when you need to. I will enjoy watching the world learn that you cannot and will not be overlooked.

Mom and Dad:

You have made this work possible in more ways than one. You have supported me through all the tough times and taught me how to learn from them. You have shown me what is important in life and the right way to live. Unfortunately, I cannot even scratch the surface of all you have given me and all I have learned from you. I hope I can live up to the example you have set. Thank you.

This work was supported by NSF grant #ITR/PE: Using Context Recognition for Preventive Medicine in the Home (#0112900) and the Changing Places / Home of the Future Consortium.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The paradigm of desktop computing is beginning to shift in favor of highly distributed and embedded computer systems that are accessible from anywhere at anytime [2]. Applications of these systems, such as advanced contextually-aware personal assistants, have enormous potential not only to benefit their users, but also to revolutionize the way people and computers interact.

1.1 Challenges of Ubiquitous Computing

The future of ubiquitous computing presents many challenges [2]. Perhaps the most difficult of these is the task of building applications and interfaces that accurately assess the context of their users. For example, a ubiquitous messaging application, perhaps running on a mobile computer, could be capable of interrupting a user anywhere and at anytime with a message, such as an email. Lacking contextual-awareness, this system would not know better than to interrupt users whenever they have a message. While possibly useful, this system would be a persistent annoyance. Instead, if the system was aware of its user's context, such as they are in a meeting with a client, and aware of which messages the user cares most about, then it would be able to interrupt users at appropriate times with important messages. Without contextual-awareness, the omnipresence of ubiquitous applications will be a deterrent to their deployment.

1.2 Contextually-Aware Systems

A significant problem related to building contextually-aware interfaces and systems is that of studying people in their natural settings, such as home or work. This problem is important because it is in precisely these environments that people will want to deploy contextually-aware applications. Researchers attempting to study people in natural settings must carefully design their studies to collect sufficient data describing a subject's context without significantly influencing their environment or behavior. The ideal balance between these factors is difficult to achieve because the strategies that are the most descriptive, such as direct observation, also tend to be the most disruptive and expensive. However, by leveraging new tools and technologies to acquire context-specific information while minimizing the burden on the subject, naturalistic and cost-effective data can be collected.

1.2.1 Context-Aware Experience Sampling Tool

The first contribution of this work is the Context-Aware Experience Sampling Tool (CAES) [23]. CAES has been built to expedite the process of developing and implementing context-aware computer applications and interfaces. It supports studies based on the experience sampling method (ESM) [13], also known as ecological momentary assessment (EMA) [37] (see Section 3.1.2). CAES is differentiated from existing ESM tools by its support for sensor data collection and context-aware sampling strategies. which are the real-time modification of ESM strategies based on sensor readings.

While a person's context could include anything from emotional to physical state, throughout this paper *context* will mean some aspect of an individual's state that can be detected by a reasonable sensor. Context-aware applications would not be able to differentiate between daydreaming and deep thought. However, such applications would be able to identify if a person is physically idle or transitioning to a new activity.

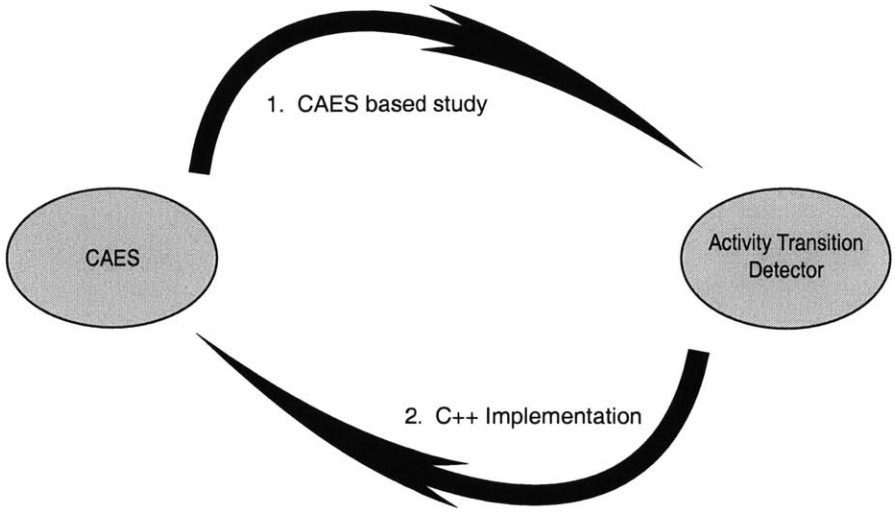

Figure 1-1: The Context-Aware Experience Sampling Tool was used to collect the data necessary to develop the activity transition detection algorithm. In future work, this algorithm could be implemented as a CAES `Interaction` class (see Section 4.1.5).

1.2.2 Activity Transition Detector

The second contribution of this work is an algorithm, developed with CAES, capable of detecting human activity transitions in office-like environments using a heart rate monitor and a planar accelerometer. The algorithm was 93.5% accurate in detecting *transitions* between sitting, walking, standing, using stairs, and lying down. This result was validated on approximately 130 hours of data collected from nineteen subjects.

The activity transition detection algorithm is an example of the type that could be deployed by contextually-aware computer systems, such as CAES, to detect appropriate times to interrupt their users. Figure 1-1 illustrates this relationship between CAES and the detector algorithm.

Chapter 2

The Problem

The first problem addressed by this research is cost-effective and non-invasive collection of qualitative and quantitative data from people in their natural settings, such as the home or office. The second problem addressed by this work is the design of technologies and interfaces that are automatically aware of good and bad times to interrupt their users.

2.1 Data Collection In Natural Settings

2.1.1 Challenges

The construction of robust context-aware interfaces required by the next generation of ubiquitous computing applications requires the collection of large amounts of qualitative and quantitative data describing the context in which the application will operate. System designers need this data to concretely characterize the user contexts and behaviors an application must be aware of. Additionally, such data is needed to train and test contextually-aware algorithms that use machine learning techniques. Collecting this data can be costly and always carries the risk that the process of data collection will significantly change the context and behaviors being studied. However, failure to thoroughly study people and their reactions to these emerging systems will ultimately undermine their commercial appeal. People will not be interested in ubiq-

uitous technologies that require constant attention or that often make the wrong assumption about the user's context.

There is no clear best option when choosing a data collection technique. In particular there are no approaches that systematically and simultaneously collect the qualitative data, such as a subject's thoughts or feelings, and quantitative data, such as heart rate.

More intrusive data collection techniques provide more detailed data but are also more likely to impact subjects' actions or environment in a significant manner. For example, subjects that know they are being observed may be careful to wash their hands after using the bathroom. However, those same subjects may occasionally forget to do so in the absence of an observer.

2.1.2 Solutions

For the desktop computing paradigm, end user data is often collected in controlled usability labs where the overall context of the user is not a consideration [27]. Ubiquitous computing system developers need an analogous tool that also considers user context.

An ideal tool would provide researchers with a low-cost automated method of collecting qualitative and quantitative data from subjects. The qualitative data would allow them to understand the needs of their users, while the quantitative data would allow them to concretely analyze the contexts of their users and build machine learning algorithms capable of detecting those contexts. The automated tool should also be easily modifiable. Researchers should be able to quickly adapt it to their study without having to learn its inner workings or make any significant modifications. Finally, the tool must be robust and easy for subjects to use. Ideally, subjects would be interrupted for self-report data only at appropriate times. If the tool cannot meet these criteria, then researchers will have difficulty finding willing subjects and collecting accurate data from users' natural settings for extended periods of time.

2.2 Designing Technology for Proactive User Interaction

2.2.1 Challenges

One of the problems central to ubiquitous computer applications is choosing a proper time to interrupt users in order to proactively present them with information. Such computer applications will be running nearly all the time and must be capable of getting their users' attention when needed. Selecting the right time and way to interrupt users can make the difference between an application that is a constant annoyance or one that is an invaluable assistant. Context-aware systems should be sensitive to users' tasks, such as when they are meeting a client, and the relative importance of the reason for interrupting.

2.2.2 Solutions

A good first step in solving this problem would be to reliably detect when users should not be interrupted. A system capable of detecting such situations would help ubiquitous system designers overcome one of the most difficult problems they face.

Chapter 3

Theory and Rationale

This research draws on previous work from the domains of activity recognition, context-aware systems, interruption scheduling, field data collection, and machine learning.

3.1 Data Collection In Natural Settings

Several methods of acquiring data from human subjects in natural settings have been developed. These can be coarsely divided into field data collection techniques and experience sampling methods.

One of the problems central to all these techniques is that they require researchers to select the behaviors of interest and a data recording schedule before the study has begun. This prevents the collection of data related to events and behaviors that researchers did not anticipate because these approaches do not allow for the dynamic modification of data collection strategies.

3.1.1 Field Data Collection Techniques

Field data collection techniques include direct observation, recall surveys, time diaries, and indirect monitoring. Each of these approaches has drawbacks that limit its capacity to support the development of contextually-aware systems.

Direct Observation

Direct observation techniques involve a researcher following a subject for the duration of a study and intermittently observing and recording the subject's behaviors [29]. The most problematic aspect of these techniques is the large amount of time and money that they require. Due to the fact that recording data prevents continuous observation, for each hour of data collected researchers must observe subjects for over an hour. Furthermore, observers must be carefully trained and tire quickly. A secondary problem with direct observation is that it introduces an observer into the subject's environment. The physical presence of another person makes it more likely that the subject's context and behavior will be effected by the study. This effect is known as reactivity.

Recall Surveys

Recall surveys present subjects with a questionnaire in an attempt to gather data on a previous event or behavior [18]. The survey's accuracy is critically dependant on the subject's ability to recall the situation being studied. The primary problems these surveys suffer from are recall and selective reporting biases. Subjects have difficulty recalling the event or behavior being studied and the information they report is often incorrect [37].

Time Diaries

Time diaries require subjects to record their behaviors as they occur or at regular intervals. This helps to minimize some of the biases that effect recall surveys [35]. However, time diaries tend to suffer from low levels of subject compliance [38] and can be quite burdensome on their users.

Indirect Monitoring

Indirect monitoring, in the form video or other sensors, addresses some of the problems of direct observation, recall surveys, and time diaries. However, indirect monitoring

must be combined with other techniques to collect qualitative data on subjects, such as feelings, reasoning, or interruptibility. As a result, indirect monitoring is often supplemented with time diaries or experience sampling [21] (see Section 3.1.2). In the absence of such supplementary data, indirect observation generally requires the researcher to review and annotate all the collected data to locate events of interest. This process can be as time consuming as direct observation.

3.1.2 Experience Sampling Methods

Overview

The experience sampling method (ESM) [13], also known as ecological momentary assessment (EMA) [37] to medical researchers, has been developed to address the deficiencies of field data collection techniques discussed in Section 3.1.1. Experience sampling is an automated version of time diaries. The least automated ESM studies require subjects to carry a diary and a device, such as a beeper, that will signal them to record data. Computerized ESM studies require subjects to carry a palm-sized computing device, such as a Palm Pilot or Pocket PC. In theses studies the computing device prompts the subject for data and electronically records their responses.

Benefits

ESM improves on time diaries and recall surveys in several key ways. First, it improves subject compliance and accuracy of the data collected [38]. Second, ESM studies reduce the burden on the subject by prompting them for information instead of forcing subjects to schedule their own recording times. Third, computerized ESM studies simplify researchers' work by reducing the preparation needed for each subject and by eliminating the intensive and error-prone data entry work that follows paper-based studies.

Problems

While ESM addresses some of the deficiencies of the field data collection techniques, it has some problems of its own. First, ESM does not provide for the collection of objective data describing the subject's behavior or context. Instead, it relies on the subject to accurately describe her own behaviors. Second, a data recording schedule, also known as a sampling strategy, cannot be responsive to the subject's environment. As a result, short activities of interest are often missed without intolerably high sampling rates.

Availability

Both commercial [24] and free [5] computerized ESM software is available. However, these existing packages are significantly less functional than the Context-Aware Experience Sampling Tool implemented in this research.

3.2 Interruptions and Activity Recognition

Research into building context-aware computer systems capable of interrupting users at appropriate times has combined work from diverse fields. Medical and activity research on sensors and sensor signal features as well as planning and machine learning techniques developed by artificial intelligence and pattern recognition researchers have found their way into context-aware systems.

Computerized ESM presents a possible application of context-aware systems research. The biggest problem ESM causes for subjects is that they are interrupted by the system at inappropriate times (see Section 5.1). To solve this problem ESM could be integrated with interruption and activity recognition systems into a contextually-aware ESM tool. Such a tool would be able to detect a subject's activity and schedule interruptions for appropriate times.

3.2.1 Interruptions

Research into how to interrupt users can be grouped into three primary categories: understanding when users are interruptible, planning interruptions, and deciding how users should be interrupted.

Deciding When to Interrupt

Work on understanding when users are interruptible has usually taken the form of human studies leveraging one or more of the techniques discussed in Section 3.1. A good example is Hudson's study [21], which shows that social interaction is a good indicator of when people are interruptible; people talking or working in groups generally do not want to be interrupted.

Research of this type could be used with the Context-Aware Experience Sampling Tool and the activity transition detection algorithm to verify the theory that people in office-like environments prefer to be interrupted when they transition between activities.

Planning Interruptions

Research on planning interruptions has focused on developing reminder strategies that are not annoying to users and ensure that tasks are efficiently accomplished [28]. Much of this work, such as Pearl [33], involves developing and testing systems that implement plan-based reminder strategies.

The interruption planning strategies developed by research in this area could be extended with the activity transition detector or other contextually-aware algorithms. This would enable interruption planning strategies capable of considering more aspects of a user's context and thus finding better interruption strategies.

Deciding How to Interrupt

The third area of interruption research has been on what content an interruption message should have to clearly communicate with users. While some novel reminder

interfaces have been developed, such as SpiraClock [15], there has not been much work addressing the core issues and challenges of reminder design [36].

The content of interruption messages is an important aspect of any complete contextually-aware application.

3.2.2 Recognizing Activities

The physical activity of the user is an important piece of context that computer applications should consider when deciding how and when to interrupt users.

Medical Activity Research

Medical and activity researchers have been exploring human activity recognition technologies for some time. Their work has focused on identifying characteristic motions of human activities, the sensors needed to identify those motions, and quantifying the energy expenditure [7, 8, 10, 17, 19].

This research is important primarily because it has validated the use of heart rate and accelerometer data to quantify and identify physical activities.

Ubiquitous Computing Activity Research

Researchers from fields unrelated to medicine, such as computer science and pattern recognition, have also studied human activity recognition. This work is more focused on developing machine learning algorithms and techniques, such as Layered Hidden Markov Models [32]. As a result, much of the work in this area relies on data gathered from a small number of subjects in limited contexts [12, 25, 39]. However, there has been some thorough work on robust activity recognition using simple sensors [34].

The relevant contribution from this area of research is the validation of machine learning techniques to identify a wide range of human behaviors from posture [25] to glances at a wristwatch [12].

3.3 Goals

The goals of this research are:

1. To improve on the existing ESM tools to better meet the needs of researchers.
2. To develop an algorithm capable of detecting good times to interrupt the users of contextually-aware computer systems.

3.3.1 Improving ESM

Existing ESM tools are improved upon by the Context-Aware Experience Sampling Tool. CAES integrates ESM features with a system capable of collecting sensor data and dynamically modifying ESM strategies based on real-time analysis of sensor and subject-provided data. In addition, CAES has been designed to be more flexible and easier to use for both researchers and subjects than existing software packages.

3.3.2 Detecting Times to Interrupt

The algorithm capable of detecting good times to interrupt users has been developed by detecting transitions between activities. When users transition between activities they are also likely to break from their current task. As a result, targeting these times for interruptions should be better than interrupting users at random times. This algorithm also presents two opportunities for CAES. First, CAES can be used to study transitions in human activities and collect data to support the development of a machine learning based detector. Second, this algorithm can be integrated back into CAES, making it an even more powerful and user-friendly system (see Figure 1-1).

A key feature of the algorithm developed is that it is highly reliable in detecting times when a user should not be interrupted. This alone is an important step toward the implementation of contextually-aware ubiquitous systems.

Chapter 4

Design and Implementation

This section presents the design and implementation of both the Context-Aware Experience Sampling Tool and the algorithm capable of detecting transitions in human activities.

4.1 Context-Aware Experience Sampling Tool

CAES is implemented in C++ and uses the Pocket PC 2002 Microsoft Foundation Classes API (MFC). This program has been tested extensively on model 3870 HP/Compaq iPAQs, but it runs on all 3800 series iPAQs. HP/Compaq iPAQs were chosen as the development platform because of their flexible input/output and sensor capabilities. Porting CAES to other Pocket PC devices will be a simple process.

Figure 4-1 shows CAES running on a 3800 series iPAQ and two other Pocket PC devices.

Figure 4-2 illustrates the interaction between primary CAES functional modules. These modules are discussed in detail throughout this section.

4.1.1 Design Criteria

The design criteria for CAES is summarized in the points below:

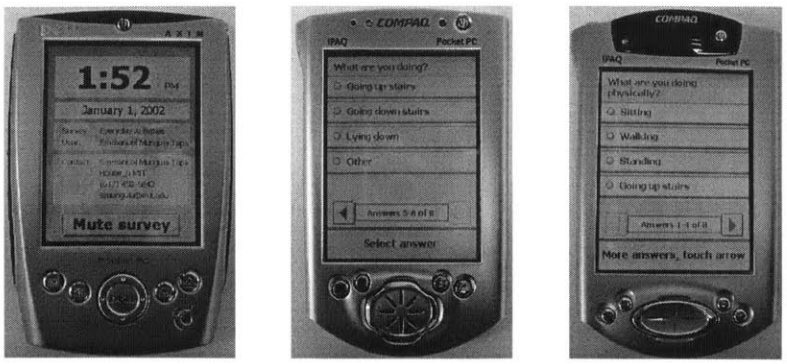

Figure 4-1: The Context-Aware Experience Sampling Tool running on a Dell Axim (left), iPAQ 3600 (center), and an iPAQ 3800 (right).

Complete Experience Sampling Functionality

CAES should be built as a next generation ESM tool. Toward that end, it should have full support of all important and popular ESM features. In the absence of sensor modules, CAES should operate as a good ESM tool.

Extensive Sensor Support

CAES should be built from the ground up to integrate a wide range of sensors for both collecting raw data and enabling context-aware ESM strategies. It should be possible to use sensors to answers questions, such as leaving an audio message or taking a picture. Using sensors should be no more difficult for researchers than specifying the survey questions and answers that will be asked of subjects. Scheduling sensor readings should be as similar to scheduling survey questions as possible.

Enhanced Flexibility

CAES should be more flexible than existing ESM tools. It should support the integration of questioning strategies, such as subject-initiated, recurring, researcher specified, and random. Each question and sensor should be independently scheduled so these strategies can be used simultaneously in the same study.

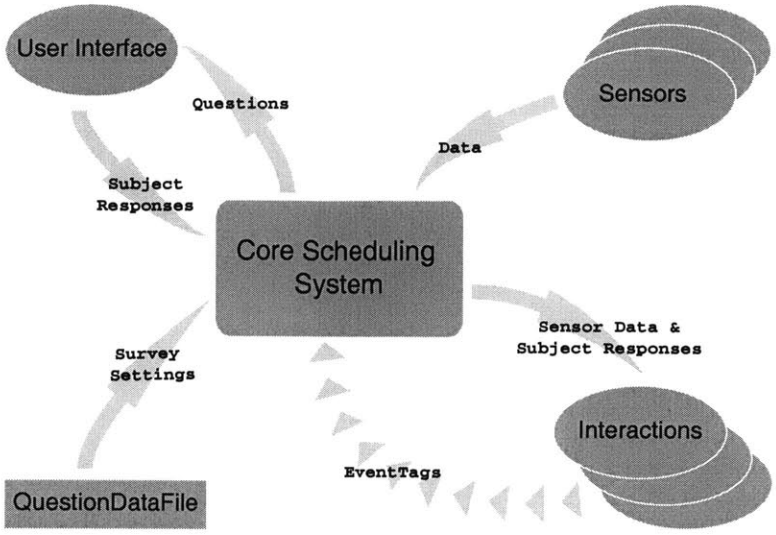

Figure 4-2: This diagram illustrates the interaction between the primary functional modules of the Context-Aware Experience Sampling Tool.

Leverage Existing Commercial Technology

CAES should be immediately usable by researchers in the real world. To achieve this it should rely on existing and commercially available technology. This applies to the hardware and software required to deploy CAES.

Easy to Start Using

CAES should be as simple for researchers to use as possible. If the system is difficult to get running, its adoption will be inhibited.

Subject Friendly

CAES should be simple and easy for subjects to use. The interface should be clean, functional, and easy to read. Subjects should not be able to access the underlying Pocket PC platform. If anything goes wrong, subjects should be able to reset the device and expect it to start working correctly. A visual help system should be implemented to walk subjects through using sensors and charging the system. Subjects should be made aware if the power runs low; however, strategies should be implemented to make it unlikely that the system will ever actually run out of power or lose data.

Extensible Implementation

The process of extending the CAES system to include new sensors and new context-aware interactions, should be painless for any reasonably experienced C++ programmer. More experienced programmers should be able quickly grasp the system architecture and make improvements if they so desire.

Modular Construction

CAES should be modularly constructed. Any major CAES functional module, such as the user interface, should be easily replaceable. CAES should be designed so that its modules could be used in the building of a new Pocket PC based context-aware system.

Open Source Distribution

CAES should be available to the research and business community to extend as they see fit. Furthermore, useful extensions to CAES should be integrated back into the distributed system.

These criteria have been met in the design and implementation of CAES.

4.1.2 User Interface

The default CAES user interface was developed over a period of several months with a professional interface designer [4] and tested extensively after its implementation. This interface has been successfully deployed with subjects eighteen to eighty years old (see Section 5.1).

The user interface has four primary screens: the `Default`, `Question`, `Device`, and `Help` screens. Figure 4-3 shows these primary screens. When not being prompted for information, the subject can interact with the `Default` screen. This screen has no functionality other than to mute the survey so that the device does not beep when the subject is in a meeting, library, or a similar setting. The `Question` screen is where the subject responds to survey questions. It currently supports both multiple choice and multiple answer questions. Additionally, the researcher may give the subject

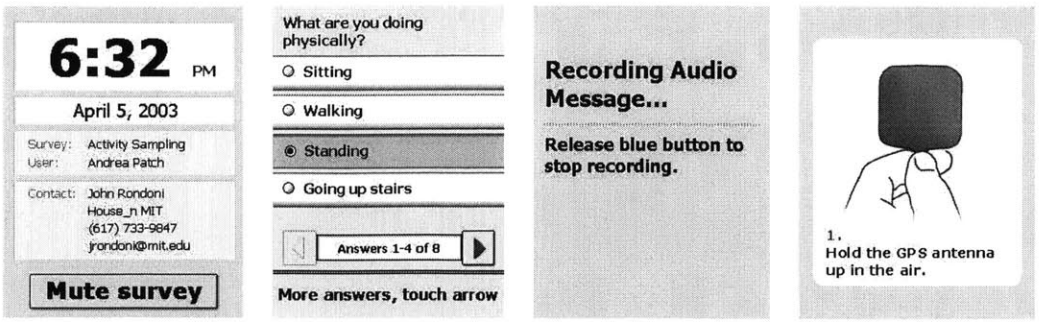

Figure 4-3: The Context-Aware Experience Sampling Tool User Interface: The `Default` screen's text (left) is customized for each study. The `Question` screen (left-center) is displaying a question from the activity transition detector study. The `Device` screen (right-center) shows the interface for recording audio messages. The `Help` screen (right) shows one of three screens that demonstrates to subjects how to use a GPS device.

the ability to answer questions by leaving an audio note or taking a picture. The `Device` screen has no active elements and is visible while readings are taken from sensors that require user interaction, such as GPS. The `Help` screen displays a series of instructions. This screen is most often used for reminding subjects how to use sensors.

The design of the user interface is focused on simplicity and usability. It is designed such that users unfamiliar with handheld computers will be able to easily interact with the system. For simplicity and security of the survey data, the interface covers the entire 320x240 pixel screen and locks out all Pocket PC functionality. The generic `Help` screen interface allows researchers to use sensors that require user interaction and to build custom help files for other purposes. For example, a researcher may use custom help files to prompt the subject to record data on a specialized paper interface, such as a map.

Due to its flexibility and success with a wide range of subjects, this user interface should meet the needs of most standard ESM and context-aware ESM studies. However, if the need arises for a new interface the existing one can be easily replaced due to the modular design of CAES.

4.1.3 Specifying New Context-Aware Experience Sampling Studies

Researchers new to CAES should be able to have the system up and running quickly. After installation, the only necessary step is to define a survey and sampling strategy in the `QuestionDataFile` (`QDF`). This can be easily accomplished using any spreadsheet tool, such as Gnumeric or Microsoft Excel, because the `QDF` is stored in a comma separated value (CSV) format.

A simplified `QuestionDataFile` is presented in Table 4.1. The first section of the `QDF` contains the survey's global settings, such as the default question, `Interaction` classes, and aggregation time (see Section 4.1.5). The second `QDF` section defines the sensors used in the study, their sampling rate, and whether the subject should be prompted to collect the data. In this example, the `HeartRate` sensor collects data continuously without subject interaction, while the GPS sensor collects data every thirty minutes and requires interaction with the subject. The final `QDF` section defines the survey questions. Questions 1 and 5 are scheduled to query the subject every fifteen minutes. The `Chain` field allows questions to be asked or sensor data to be collected in response to subjects' responses. In this example, question 30 is asked only when the subject responds to question 1 with *Reading*. Similarly, the GPS sensor is sampled when the subject responds to question 5 with *Yes*.

If a previously unsupported sensor is required, the researcher will need to program a new `Sensor` class (see Section 4.1.4). If a new context-aware capability is needed, the researcher will need to program a new `Interaction` class (see Section 4.1.5). Most researchers will be able to use CAES with no programmatic modifications.

To illustrate the flexibility and capabilities of CAES, consider a psychology study of human emotions. To study people's emotions on a daily basis a researcher may want to prompt subjects' when their heart rate significantly increases in the absence of physical activity. To detect heart rate the researcher may decide to use a Polar heart rate monitor and to control for physical activity the researcher may decide to fit subjects with an accelerometer. This study would be able to leverage the

Keys	DataTypes	Values
SurveyName	*String*	Example QDF File
DefaultQuestion	*ID*	5
Aggregation	*H:M*	0:10
QuestionTimeOut	*H:M*	0:2
Interactions	*ClassString*	HeartRateInteraction
ForceAggregation	*EventTagString*	HEARTRATE_INCREASE
...	...	...

ID	DeviceType	UserSample	Continuous	Recur	Chain	...
>0	*DeviceString*	*Y/N*	*Y/N*	*W:D:H:M*	*ID*	...
100	SerialGPS	Yes	No	0:0:0:30	1	...
200	HeartRate	No	Yes			...

ID	QuestionType	Recur	Question	Answer	Chain	...
>0	*QuestionString*	*W:D:H:M*	*String*	*String*	*ID*	...
1	MLTPL_CHCE	0:0:0:15	What are you doing?	Reading	30	...
5	MLTPL_CHCE	0:0:0:15	Are you outside?	Yes	100	...
30	MLTPL_ANSR		What do you read?	Books		...

Table 4.1: a Simplified CAES QuestionDataFile Example. The top table contains the global settings for a survey. The center table contains the survey sensor definitions. Only sensors that are used in a study need to be defined. The bottom table contains the question definitions.

existing Polar heart rate monitor Sensor class. However, a new Sensor class for the accelerometer and a new Interaction class to detect emotional events will have to be programmed.

Once the programming is complete, CAES would be capable of monitoring subjects' heart rate and physical activity. When the researcher's new Interaction class detects an accelerated heart rate in the absence of physical activity it would signal the core scheduling system by sending an EventTag (see Section 4.1.5). The core scheduling system would then take action based on the researcher's QuestionDataFile. In this case, the EventTag would cause the subject to be prompted with questions concerning their emotional state.

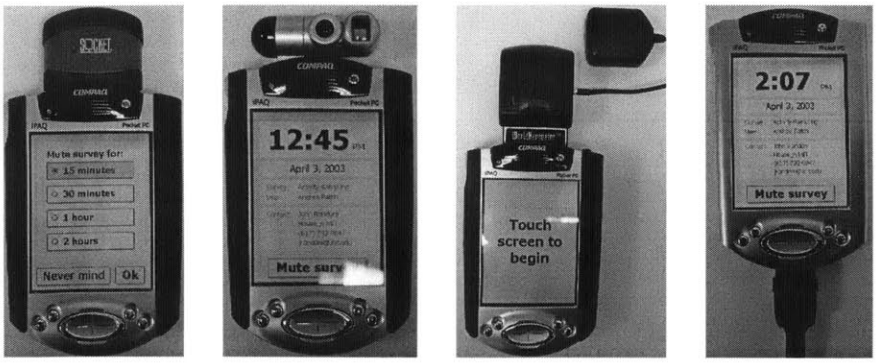

Figure 4-4: These images show the Context-Aware Experience Sampling Tool with sensors: The bar code scanner (left) can read standard 2D bar codes. The camera (left-center) enables subjects to take a picture as an alternate answer to questions when enabled by the researcher. The GPS receiver (right-center) provides outdoor location data. The iPAQ serial cable (right) enables CAES to support for a wide range of sensors, such as a wireless Polar heart rate monitor.

4.1.4 Sensor System

The CAES sensor system is designed to support a large number of sensors, such as GPS, heart rate monitor, bar code scanner, and microphone. Figure 4-4 shows several sensors CAES currently supports. Only a small subset of these sensors can be used simultaneously due to limited input/output capabilities of a Pocket PC. CAES takes advantage of this and only creates internal instances of those sensors defined in the `QuestionDataFile`. Sensors in CAES have two roles: the collection of data and the context-sensitive triggering of subject interaction. Sensors can serve either of these two roles or both simultaneously.

The CAES sensor system was specifically designed to be easily extensible as new sensors become available. Programmers are provided with two tools to aid in extending the CAES sensor capabilities: the `DataSource` and `Sensor` classes.

Data Sources

`DataSource` classes encapsulate a low level input stream, such as a serial or network port. This allows programmers, even if they are unfamiliar with Pocket PC APIs, to quickly integrate new sensors that rely on standard communication methods. By

leveraging `DataSource` classes, a programmer can access new sensors in three lines of C++ code: one line to create an instance of the appropriate `DataSource` class, one line to initialize the class by calling `Initialize()`, and one line to get data from the sensor by calling `GetBytes(BYTE* buf, UINT length)`.

Sensors

All sensors supported by CAES extend the abstract `Sensor` class. The `Sensor` class greatly eases the burden on the programmer integrating a new sensor into CAES. `Sensor` implements all the thread protection and core scheduling system interfacing functionality that a sensor requires. Programmers adding a new sensor to CAES need only to extend the `Sensor` class and implement its six abstract functions. Because all the CAES system code has already been implemented, programmers of new sensors can focus on the sensor specific functionality.

By leveraging both the serial port `DataSource` implementation and the `Sensor` class, a Polar heart rate monitor was added to CAES in 41 lines of new C++ code. Seventeen of those 41 lines of code are used only for storing the time-stamped heart rate data to a tab-delimitated file.

4.1.5 Interaction System

The interaction system implements CAES contextual-awareness and context-sensitive triggering capabilities. The interaction system has two primary components: `Interaction` classes and the core scheduling system's list of `EventTags` and associated actions. `EventTags` are a secondary component of the interaction system. They are the messages that link the core scheduling system and the `Interaction` classes.

Interaction Classes

Before any sensor or subject-provided data is stored it is passed through all `Interaction` classes specified by the researcher in the `QuestionDataFile`. At this point each class can perform any processing on the data. The `Interaction` classes act as detectors.

When they notice an interesting event in the sensor or subject-provided data they can alert the core scheduling system by sending an `EventTag` descriptive of the detected feature.

Core Scheduling System

A list linking `EventTags` to actions is maintained by the core scheduling system. These links are created according to the researcher's specification in the `QuestionDataFile`. When a particular `EventTag` is received, one of several things can happen. First, `EventTags` can be linked to questions or sensors, causing them to be activated, deactivated, or triggered. Triggered questions prompt the subject for information, while triggered sensors gather data. Second, `EventTags` can be linked to system events and cause CAES to terminate execution or cause all timers to be aggregated. When all timers are aggregated, any questions or sensors scheduled to be triggered within the `QuestionDataFile` specified aggregation period will be triggered.

Extending the Interaction System

The interaction system is central to the contextually-aware functionality of CAES. It is expected that researchers will need to extend the interaction system to implement new context-aware capabilities. To support this, programmers have been provided with an abstract `Interaction` class that implements the necessary system interfaces and thread protection. As a result, the programmer is free to concentrate on the signal processing and context-aware functionalities that their new class must implement. Programmers of new interactions are required only to implement the `RunInteraction()` function, which is where all signal and data processing should occur.

4.1.6 Core Scheduling System

At the heart of CAES is a timing and scheduling system. This system is configured on each run by a researcher specified `QuestionDataFile` (see Section 4.1.3). This

file contains the questions that are to be asked of the subject as well as the sensors and sampling strategies to be used. The core scheduling system is responsible for notifying the user interface when the subject is to be prompted for information. It also works with the user interface to manage the system's power. The core scheduling system enables CAES's context-awareness by maintaining a list of `EventTags` and a list of actions to be taken when each `EventTag` is received. In addition, this system routes sensor and subject-provided data through active `Interaction` classes before the data is stored (see Section 4.1.5).

The core scheduling system should not require customization beyond that provided by the `QuestionDataFile`.

4.1.7 Status of CAES Project and Software

The Context-Aware Experience Sampling Tool has been implemented as described in this paper and is freely available as MIT Licensed Open Source Software [11]. This project is being maintained by researchers at the MIT House_n / Home of the Future Project [30].

4.2 Activity Transition Detector

The activity transition detection algorithm has been designed for eventual implementation as a CAES `Interaction` class. This would allow CAES to target questions at activity transitions. Targeting interruptions for activity transitions should be better than interrupting users at random times because users are likely to be taking a break from their current task when they transition between activities.

4.2.1 Activity Transition Selection

For tractability, the general problem of detecting transitions in human activities was narrowed to focus on those activity transitions that occur in office and classroom-like environments. For the purposes of this study, office and classroom activities include:

sitting, standing, walking, using stairs, and lying down. These activities were selected because they have been the subject of successful activity recognition studies based on simple sensors [14, 17, 26, 34] and because transitions between them should be generally indicative of changes in a person's task. For example, when a person gets up from her desk to get a cup of coffee, she takes a break from desk work and undergoes a transition in activity from sitting to walking.

4.2.2 Sensor Selection and Placement

For developing the activity transition detector a Polar [16] wireless heart rate monitor and a planar accelerometer were chosen. Previous medical activity and pattern classification research shows that these sensors are sufficient to recognize the selected activities [14, 17, 34]. This implies that those sensors should also be good choices for detecting the transition between activities.

Heart Rate Monitor Selection

The heart rate monitor was selected because it is a good indicator of transitions between postures, such as sitting and standing [10, 17]. Furthermore, the heart rate monitor may also indicate transitions between static and dynamic activities, such as sitting and walking, or between dynamic activities that require different levels of exertion, such as walking and using the stairs. Figure 4-5 shows the Polar heart rate monitor worn by subjects as well as the circuit board used to receive the heart rate. The receiver sent its data over a short wire to the iPAQ serial port.

Planar Accelerometer Selection

The accelerometer was selected for three primary reasons. First, a planar accelerometer can indicate posture and therefore differentiate between different static activities, such as sitting and standing still [17]. Second, the accelerometer signal should distinguish between static and dynamic activities [10]. Third, accelerometers have been shown to capture subtle changes in patterns of motion that may help differentiate

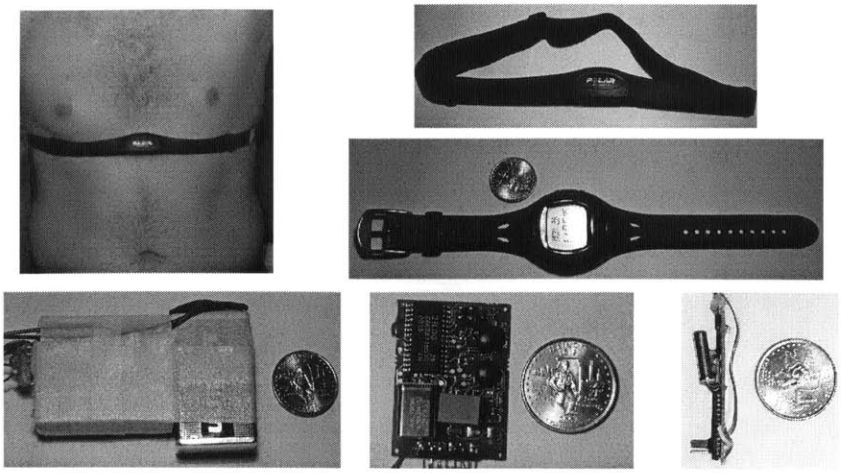

Figure 4-5: the heart rate monitor. The top three images show the standard commercial Polar wireless heart rate monitor configuration. The heart rate monitor is worn around the center of the torso (top-left). The heart rate monitor transmits the heart rate wirelessly to a receiver in a watch that displays the users' heart rate (top-right). The bottom three images show the custom heart rate receiver that was connected to the iPAQ for this study. The heart rate receiver board is small enough to be comfortably worn in subjects' pockets or on their belt (bottom-center & bottom-right). The board needs its own power supply in the form of a nine-volt battery, but is still reasonably sized (bottom-left).

between dynamic activities such as walking and using the stairs [3, 20].

Planar Accelerometer Placement

For this study, the planar accelerometer was placed on subjects' outer left thigh; tangent to their side such that the Y-axis runs vertically along the subjects' body and the X-axis runs horizontally from the subjects' back to their front.

The accelerometer was thigh-mounted for several reasons. The accelerometer board being used, shown in Figure 4-6, was bulky and uncomfortable for extended wear in many places. For example, if placed on a subject's back, they would not be able to sit normally. The ADXL202 accelerometer used is only capable of measuring up to two G's, while the extremities of human limbs receive nearly ten G's of acceleration in many activities [7]. As a result, limb extremities were not a good placement choice. In addition to these criteria, the thigh was chosen because it would

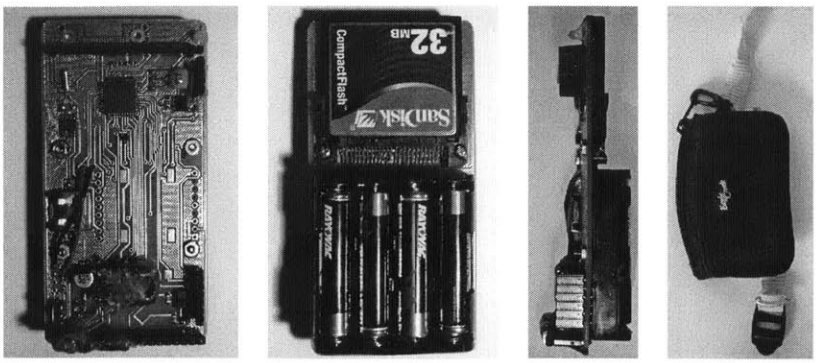

Figure 4-6: the Planar Accelerometer. The accelerometer is mounted on the top of the data collection board (left). The board is powered by four AAA batteries and data is stored to the compact flash card located on the underside of the board (left-center). The board is rather thick (right-center); however, it was easily mounted on subjects' outer left thigh using a coin purse and elastic strap (right).

allow the planar accelerometer to differentiate between standing, when the X-axis is perpendicular to gravity, and sitting, when the Y-axis is perpendicular to gravity.

4.2.3 Survey and Sampling Strategy

Subjects were asked seven questions by CAES for this study. Six of the questions were used to identify activity and one addressed how open to interruption the subject was. In any one question series a subject would have to answer at most three questions: two for identifying their activity and one concerning how willing to be interrupted they were. These questions were presented to the subject at regular intervals of four minutes. This sampling rate was chosen because it was the highest sampling rate preliminary testers found tolerable for studies up to eight hours.

To increase the number of activity labels near activity transitions, a simple context-aware `Interaction` module was developed. This module would send an `EventTag` when a subject's heart rate rose or fell continuously for fifteen seconds or when it changed by twenty percent or more in a second. The `QuestionDataFile` specified that this `EventTag` would aggregate questions and prompt the user for an activity label (see Section 4.1.5). Figure 4-7 shows an example of where this context-aware

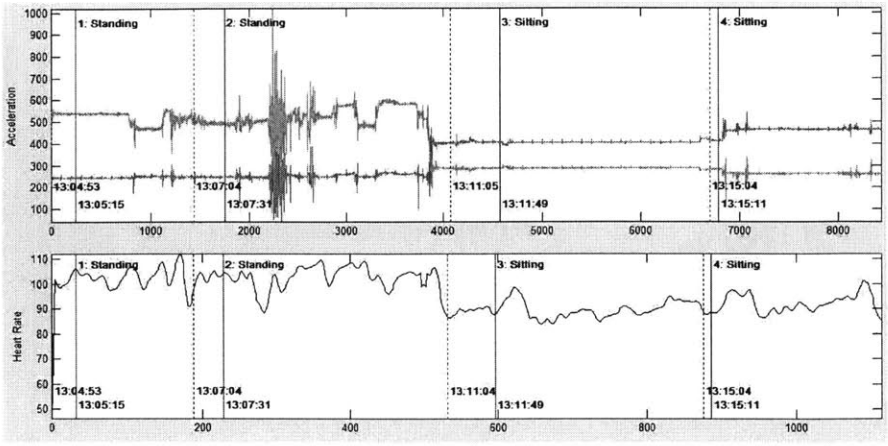

Figure 4-7: These plots show an example where a change in heart rate over twenty percent caused CAES to prompt the subject for an activity label. The dotted line indicates when a subject was prompted for the activity label. The immediately following solid line indicates when the subject responded and the activity label they provided. As a result of prompting the subject for this information, CAES was able to accurately label an activity transition from standing to sitting. See subject-provided activity labels two and three in the above image.

triggering occurred and improved the accuracy of the data.

4.2.4 The Labeling Problem

The labeling problem is the challenge of accurately identifying all activity transitions using CAES. Two labeling difficulties arose during this study. First, subjects occasionally entered incorrect information. Second, in some cases subjects underwent two symmetric activity transitions, such as sitting to standing and standing to sitting, between providing activity labels. As a result, a transition occurred but was not labeled.

To allow researchers to evaluate the effectiveness of subject labels and to modify them when they are obviously wrong, a Matlab interface capable of viewing an entire day of planar accelerometer and heart rate data along with subject-provided labels was developed. The interface allows researchers to view the data by zooming in and out as well as scrolling temporally. The subject-provided activity labels can be modified or deleted and new activity labels can be added simply by clicking on the

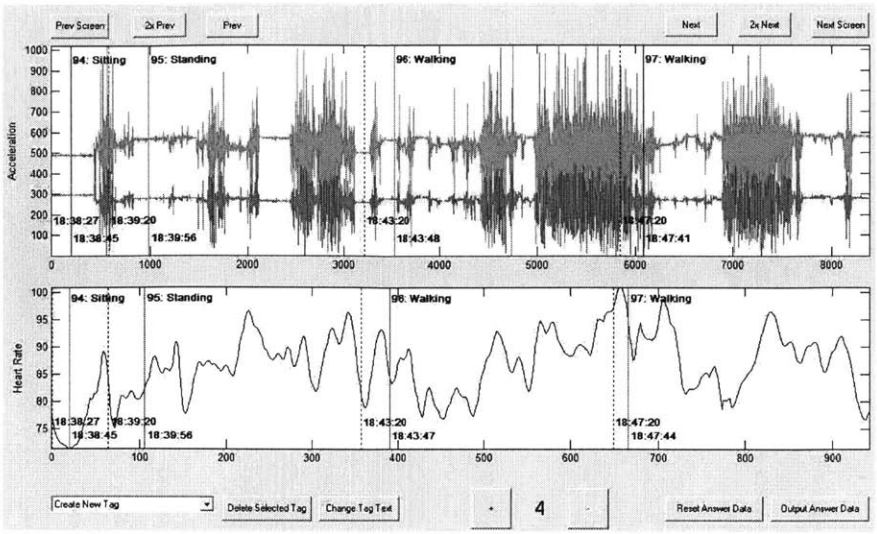

Figure 4-8: This tool displays the planar accelerometer and heart rate data. The user is able to scroll temporally with the controls at the top of the interface. The lower controls allow the user to modify existing activity labels or add new ones. Additionally the user can zoom in and out to view different time periods.

graphed data. Figure 4-8 illustrates this interface and its functionality.

Section 5.2.2 demonstrates the use of this tool to evaluate the extent of the labeling problem.

4.2.5 Representation

State Space Description

Most machine learning human activity recognition research relies on a representation that describes subjects' activities as a series of states. For example, consider a timeline annotated with subjects' activities, such as walking and running. These models of human activity usually return the likelihood that an individual is performing a particular activity at a given time. Such an approach is appropriate if the problem to be solved is determining the current activity of an individual. However, this is not the problem being addressed by the activity transition detector. This algorithm is attempting to detect the *transitions* between activities, not the activities themselves.

Transition Space Description

Borchardt's Transition Space representation is based on the insight that the correct way to reason about causal descriptions and temporal events is often not as a series of states, but as a series of transitions [6]. Consider a car accident. A State Space description of an accident would be: The cars were moving. Next they collided. Then they stopped. In contrast, a Transition Space description would be: The distance between the cars decreased. Next the distance between the cars disappeared. Then the velocity of the cars decreased and disappeared. The important difference between these two descriptions is that state based one focused on a series of states while the transition based one focused on the important changes between the states.

Transition Space Benefits

The Transition Space paradigm is a better choice for a detector of transitions between human activities for several reasons. First, the transition space representation focuses the designer's attention on the features that describe transitions instead of those that describe activities. While some features may be descriptive of both activities and transitions between them, this is not always the case. For example, the Transition Space representation suggests features, such as change in heart rate, that are not descriptive of a particular activity but are descriptive of transitions between them.

Leveraging the Transition Space representation allows the actual activity of a subject to be ignored. The advantage of this is that it enables the detection of transitions between activities not anticipated by the system designer. An activity recognition or State Space approach can only detect transitions between activities it is able to recognize. At best, a well designed State Space system would also be aware of when an unknown activity is being performed. However, such a system would be helpless to detect transitions between unknown activities. In contrast, the Transition Space approach allows these types of unknown activity transitions to be detected because the system is not burdened with having to first detect individual activities before detecting transitions between them. This also allows the detection

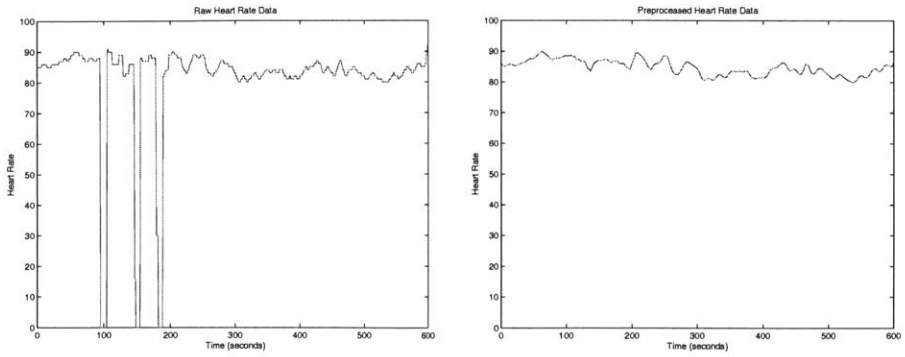

Figure 4-9: These figures shows the same ten minutes of heart rate before and after it was preprocessed. The zeros in the raw data (left) are the result of a lost wireless connection. The missing heart rate values were computed by averaging the nearest good data points. As a final step, the preprocessed signal (right) is low pass filtered.

of transitions between hard to detect activities.

4.2.6 Data Preprocessing

This section describes the preprocessing steps taken with the heart rate, planar accelerometer, and subject-provided activity labels collected by CAES. Following the preprocessing, features are computed on the data (see Section 4.2.7) and the features are used in machine learning algorithms to complete the activity transition detector's development (see Section 4.2.8).

Heart Rate Monitor

The heart rate monitor data was noisy due to the wireless connection and occasional poor readings.

Two steps were taken to clean the heart rate monitor signal. First, all unreliable heart rate readings and those that differed by over 25% from the previous reading were replaced with the average of the first previous and following good data points.

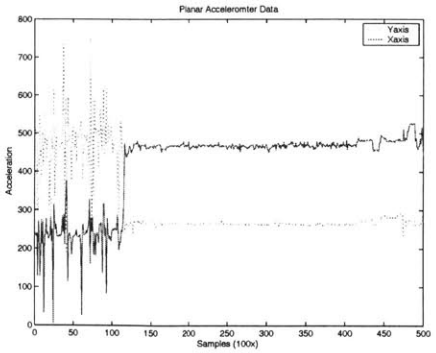

Figure 4-10: This is a plot of ten minutes of planar accelerometer data. The crossover of the X and Y-axis signals is indicative of a change in posture from standing to sitting. The planar acceleration is measured on a scale from 0 to 1023; a value of 512 indicates zero acceleration, 1023 indicates 2G's of acceleration, and 0 indicates -2G's of acceleration.

Second, the signal is low pass filtered such that:

$$y[n] = \frac{1}{9} \sum_{i=n-8}^{n} x[i].$$

Each value of the filtered signal is the average of it and the eight previous unfiltered values. This process was recommended by Polar, the heart rate monitor manufacturer [22, 9], and yields a cleaner and more reliable signal. Additionally, this filter requires very little computation and thus can be easily implemented in a real-time algorithm. Figure 4-9 illustrates the effectiveness of heart rate data preprocessing with ten minutes of data from this study.

Planar Accelerometer

The planar accelerometer data collected from subjects' left thighs required no preprocessing. Figure 4-10 shows ten minutes of accelerometer data collected for this study.

Subject Question Responses

The only noise in the question data was a result of subjects entering incorrect data. While this is a concern, no pre-processing techniques can effectively account for this. The best that can be done is to compare the subject-provided activity labels to the heart rate and planar accelerometer data to ensure that the subject's responses were generally accurate. This problem is discussed more generally in Section 4.2.4. The extent of the incorrect subject-provided data is evaluated in Section 5.2.2.

4.2.7 Features

This section describes the features that were calculated on the planar accelerometer and heart rate data. All of these features are based on the temporal change of an interesting characteristic in the raw data. This is a direct result of the Transition Space approach, which emphasizes directly identifying transitions between activities by analyzing temporal changes in the data.

Heart Rate Monitor Features

Only two features were computed on the heart rate data: the change in mean heart rate and the change in the average slope of the heart rate.

Change in mean heart rate was chosen because different physical activities cause human hearts to operate at different rates. As a result, changes in mean heart rate may be indicative of transitions in activity.

Change in the heart rate slope was chosen because human heart rates do not make discrete jumps. Instead the heart rate changes gradually as people transition between activities. Additionally, human heart rates tend to remain steady when a person's activity stays the same. For these reasons, significant changes in the slope of heart rate may be indicative of transitions in activity.

Planar Accelerometer Features

Four features were computed on the planar accelerometer data: change in mean X and Y-axis acceleration, change in the mean difference between X and Y-axis acceleration, change in the primary frequencies of the X and Y-axis signals, and change in the magnitude of the primary frequencies of the X and Y-axis signals.

Change in the mean accelerometer values was selected as a feature because those values are indicative of posture. For example, when a subject is standing, the Y-axis is parallel to gravity's acceleration while the X-axis is perpendicular to it. A large Y-axis acceleration and a small X-axis acceleration is descriptive of a standing posture. Changes in posture are important because significant changes in posture should also be indicative of transitions between activities.

Change in the mean difference between the X and Y-axis acceleration was also selected because it is indicative of posture. A large mean difference between the accelerometer signals is descriptive of the fact that the larger of the two is closer to parallel with gravity while the smaller is closer to perpendicular with gravity. A small mean difference between the signals is descriptive of a posture in which both the X and Y-axis accelerometers are offset from vertical by a similar degree. As a result, significant changes in the mean difference between X and Y-axis accelerometer signals may indicative of transitions between postures and activities.

Change in the primary frequencies and their magnitude should be indicative of changes between dynamic activities, such as walking and using stairs. The frequency of human leg motion is different for walking up stairs, walking down stairs, and walking on flat terrain. Therefore, significant changes in primary frequencies and their magnitude should be indicative of transitions between activities.

Calculating Features

Calculating the features from the pre-processed signals is a three step process.

1. *Creating and Labeling Data Segments*

   First the pre-processed data is segmented based on the subject-provided activ-

ity labels. Each data segment contains heart rate and accelerometer data for the period between two subject-provided activity labels.

After the data segments are created they are labeled as *transition* or *stable*. Segments with the label *transition* have different subject-provided activity labels directly preceding and following them. Segments with the label *stable* have identical subject-provided activity labels directly preceding and following them.

2. *Calculating Feature Precursors*

   Each of the features described has a pre-change precursor, such as mean heart rate for the change in mean heart rate feature and mean X-axis acceleration for the change in mean X-axis acceleration feature. These feature precursors are calculated for each *static* data segment.

3. *Segment Comparison*

   The final features are calculated by comparing the feature precursors of different *static* data segments.

Each *static* segment is compared to the first two of the three segments that temporally precede it, if those segments are also labeled *static*. Segments are not compared to their immediate predecessor because an activity transition takes a significant amount of time. The segments temporally located between those compared is where an activity transition may have occurred.

After the final features are calculated by comparing the feature precursors of two segments, each segment comparison is labeled. If a *transition* segment is temporally located between the two *static* segments compared, then the comparison is labeled a *transition*. If all the segments temporally located between those compared are labeled *static*, then the comparison is labeled *static*. Figure 4-11 illustrates how this process works.

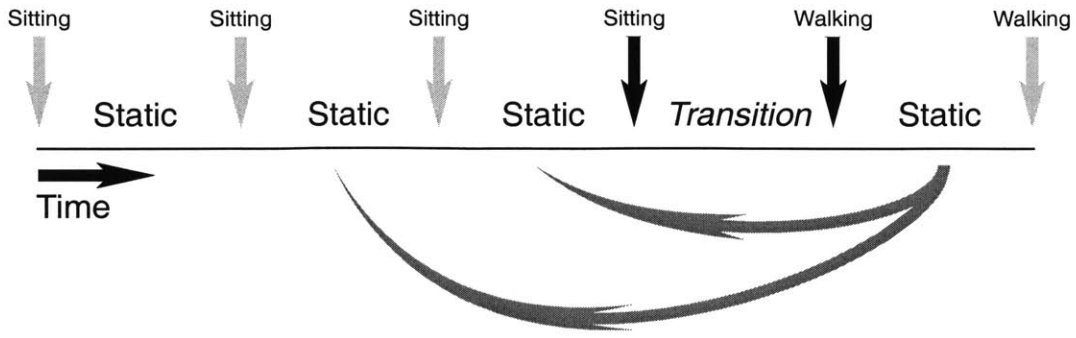

Figure 4-11: This illustrates the process of comparing feature precursors of each *static* segment to the feature precursors of first two of the three segments that temporally precede it. In this case, the first two of the three segments that precede the current one are both *static*. As a result, the current segment is compared to both. There is a *transition* segment located between those compared. Therefore, both these comparisons with their features will be labeled as *transition* instances.

4.2.8 Machine Learning

The final step in constructing the activity transitions detector was to build and test a classifier. Standard machine learning techniques were used to build classifiers based on the segment comparisons. The performance of all classifiers were verified with ten fold cross-validation.

Section 5.2.4 presents the performance of the machine learning algorithms. The WEKA [1] machine learning tool was used to perform the machine learning tasks.

Chapter 5

Evaluation

This section aggregates the feedback received concerning CAES and the algorithm for detecting activity transitions.

CAES feedback has come from both researchers and subjects involved with CAES based studies. Feedback was received from a broader community at the 2003 CHI conference [23] and through more informal channels.

The activity transition algorithm has been evaluated primarily on the basis of its performance on approximately 130 hours of data collected from nineteen subjects.

5.1 Context-Aware Experience Sampling Tool

Evaluation of CAES has primarily been a process of eliciting subjective feedback from researchers and subjects using the system.

5.1.1 Patterns of Daily Life Study

CAES is currently being used in an ongoing MIT study into the patterns of activities people perform in their own home. In this study CAES is being used to ask subjects to identify their current activity, such as watching TV or cooking. The subject self-report labels will then be synchronized with data collected from a large number of simple switch sensors taped throughout the subjects' homes [31]. At the time of

writing, two subjects in this study had used experience sampling for sixteen days each. The first subject was a thirty year-old female and the other was an eighty year-old female.

At the conclusion of the studies, each of these subjects was interviewed and asked about the study and CAES. In both cases the subjects indicated that the experience sampling was disruptive to their activities. They also both commented that the CAES requirement that they view all possible answers before completing a question was annoying. However, most of the comments from both subjects concerning CAES were focused on the questions and answer choices the researcher chose to present them with.

At one point during the sixteen day study with the first subject, CAES failed to wake-up and start prompting the subject for information in the morning. At the conclusion of the study the CAES log file was studied and a minor bug that was the likely cause of this problem, was discovered and fixed. This behavior has not been observed since this fix was made.

An interview before the start of the study with the eighty year-old subject revealed the need for a modification to CAES. This subject had difficulty reading the question and answers quickly and thus needed longer than normal to answer each question. CAES, however, would begin to beep at the subject only 90 seconds after presenting them with a question. To solve this problem a line was added to the `QuestionDataFile` to allow researchers to specify the delay before CAES starts beeping at the subject and before CAES assumes the subject has ignored the question.

Overall the results of these studies were positive for CAES. Subject frustrations were not with the tool, but with the interruptions inherent to all ESM studies. Only two modifications of the CAES software were required. One modification added an important feature and one fixed a minor bug. It is a testament to the CAES interface and design that most of the subjects' comments focused on the researcher's survey design. This implies that the software was unremarkable, easy for the subjects to use, and that the success of a deployment will be determined by a researcher's survey and sampling protocol, not by CAES. These studies are confirmation that CAES

can be successfully deployed for extended periods of time without any researcher intervention.

The end of study interviews with these two subjects suggest a few guidelines for researchers as they design and deploy CAES based surveys. First, each question should be as easy to understand and answer as possible. Second, overall survey organization and question ordering should be carefully considered. For example if the subject indicated five minutes ago that they were watching TV, the survey should assume they are still watching TV. If that is not possible, the survey should at least ask if the subject is still watching TV before requiring them to choose from a long list of possible activities. Finally, whenever possible questions and their answers should be tailored to each subject's unique situation.

5.1.2 Activity Transitions and Interruption Study

At the conclusion of the study with each subject, a brief interview was held to gather the subject's feelings about the study and CAES.

There were several comments that echoed throughout most of the interviews. The first was that a four minute CAES sampling rate was at, if not beyond, the limit of what is tolerable for a business day. Subjects also generally disliked having to view all the answers to a questions before making their selection. Finally, when directly questioned about the software user interface, subjects responded that they found it intuitive and easy to use.

It is interesting to note that subjects quickly became aware of the context-aware question triggering used in this study. Nearly every subject mentioned that CAES seemed to prompt them for activity labels when they transitioned between dynamic and static activities, such as walking and sitting.

Throughout this study CAES was reliable. No changes to the software were required. Some modifications to the survey and sampling strategy were required after preliminary testing.

5.1.3 Additional Studies and Interest

Discussions about the capabilities of CAES with other researchers were quite positive. These conversations took place in the context of the 2003 CHI conference [23] on human-computer interfaces and also in more informal discussions. Several researchers from other universities indicated they were interested in using CAES in their own work. Since CAES is a highly flexible and freely available tool [11], it should prove applicable to a wide range of ubiquitous system, context-aware interface, and human behavior research.

5.2 Detection of Activity Transitions

The evaluation of the algorithm capable of detecting transitions between human activities is highly objective compared to the evaluation of CAES.

5.2.1 Subjects

To properly evaluate the activity transition detection algorithm it was necessary to collect real-world data from a significant number of subjects. A major pitfall that many ubiquitous computing and context-aware system research projects fail to avoid is verification of their results with data from real people doing real things. These systems are too often evaluated in laboratory settings with the only subjects being the same researchers that implemented the system.

For this work, data was collected from MIT students, students from other Boston-area colleges, MIT researchers, MIT staff, and a Boston area lawyer. Table 5.1 contains additional demographic data on the subjects.

5.2.2 Data

Approximately 130 hours of heart rate data, planar accelerometer data, and subject-provided activity labels were collected from the nineteen subjects. The mean study

Subject Sex

Number of Female Subjects	3
Number of Male Subjects	16

Subject Age

Oldest Subject	33
Youngest Subject	18
Mean Subject Age	23

Table 5.1: Activity Transition Detector Subject Demographics

length per subject was 6.8 hours; with a minimum of 4.5 hours and a maximum of 10 hours.

5.2.3 The Labeling Problem

The Matlab interface for viewing subject data and correcting activity labels was used to critically evaluate the extent of the labeling problem (see Section 4.2.4). Two subjects' data and activity labels were evaluated for correctness. While it was not possible to determine the correctness of all subject-provided activity labels, it was possible to do a sanity check on them. Consider the following scenario: the Y-axis of the planar accelerometer is measuring gravity's acceleration and the X-axis is measuring little or no acceleration. At such a time the subject is almost certainly standing; a `SITTING` activity label would be clearly wrong. In addition, significant symmetric activity transitions, such as sitting to standing and standing to sitting, occurring between activity labels can also be properly identified and labeled.

Overall the subject-provided activity labels were accurate. Most of the errors were the result of symmetric activity transitions between user provided labels. The first subject evaluated had only four unlabeled transitions during a seven hour study, all of which were symmetric activity transitions between activity labels. The second subject had ten unlabeled transitions in a ten hour study, eight of which were symmetric. Only the second subject provided answers that were obviously incorrect. There were two such answers provided by the second subject; however, in both cases the subject transitioned between activities while answering the question. It seems that these incorrect answers were accidental.

From this analysis it is clear that one modification to the study would catch most of the transitions missed: prompting the subject for activity labels when X and Y-axis accelerometer signals undergo significant and simultaneous changes.

5.2.4 Performance

The performance of the activity transition detection algorithm based on a Naive Bayesian and C4.5 classifier is presented in Table 5.2. These performance characterizations are based on ten-fold cross-validation of the algorithms. The percentages above the tables indicate the overall percentage of correctly classified instances. The table rows represent the correct classification and the columns contain the algorithm provided classification. The top two tables contain the likelihood of a *static* or *transition* class receiving a particular classification. The bottom two tables contain the number of instances from each class receiving a particular classification.

The Naive Bayesian classifier is good at correctly detecting activity transitions. However, because it misclassifies a significant number of *static* instances as *transition*, the Naive Bayesian approach causes a high number of false positives. In this case, only 59.5% of the instances classified as *transition* were actually transitions. This is a problem because if actually deployed, the Naive Bayesian classifier would only be correct about 60% of the time when it determined that a user was transitioning between activities. As a result, users would be interrupted incorrectly 40% of the time.

Overall the C4.5 classifier is the better choice. While it fails to correctly detect a larger number of activity transitions, it is better at detecting *static* instances; the absence of an activity transition. Due to the much larger number of *static* instances, the C4.5 based classifier has a much lower false positive rate: 17%. Although the actual number of instance classifications that are different between the Naive Bayesian and C4.5 classifier is relatively low, C4.5 is significantly more reliable when it classifies a instance as *transition*. This classifier will interrupt users correctly 83% of the time.

Naive Bayesian: 89%

Static	Transition	
92%	8%	*static*
30%	70%	*transition*

C4.5: 93.5%

Static	Transition	
98%	2%	*static*
32%	68%	*transition*

Static	Transition	
1808	155	*static*
96	227	*transition*

Static	Transition	
1919	44	*static*
103	220	*transition*

Table 5.2: This table contains the classification results of the Naive Bayesian and C4.5 algorithms. The percentages above the tables indicate the overall accuracy of each classifier. The rows of all the tables represent the correct classification, while the columns represent the resulting classification. The top two tables show the likelihood that an instance of a particular class will receive a particular classification. The bottom two tables show the actual number of instances of each class that received a particular classification.

5.2.5 Analysis

The activity transition detection algorithm is very reliable detecting the lack of transitions between activities and is still reasonably sensitive to activity transitions.

This algorithm was developed to help the designers of context-aware ubiquitous computer systems identify better times to interrupt users. If further research shows that it is better to interrupt people when they transition between activities instead of a random time than this algorithm is a success.

This algorithm is very effective at detecting when people are not transitioning between activities and so would be highly accurate in detecting when not to interrupt users. This is exactly what is desired, a reliable method of detecting when a users should not be interrupted.

While this activity transition detection algorithm is more effective at detecting when people do not transition between activities, it is still a reasonably sensitive detector of transitions. More importantly, it is a reliable detector of transitions. A *transition* classification is highly reliable with a C4.5 based system.

Chapter 6

Conclusions

There are two primary contributions of this work: the design and implementation of the Context-Aware Experience Sampling Tool and the development of an algorithm capable of detecting real-world human activity transitions.

6.1 Context-Aware Experience Sampling Tool

CAES has proven itself, in testing and several studies, to be a robust and useful tool for the study of natural human behaviors and contexts. This tool provides ubiquitous computing researchers and context-aware systems researchers with a cost effective and flexible way to study people and collect data for the implementation of new systems and interfaces.

The data collected by CAES in this and other studies will allow researchers from a wide range of disciplines to analyze human behavior in natural environments.

While CAES is fully implemented and freely available, future work could focus on the development of more advanced `Interaction` classes and the integration of new sensors into the tool.

6.2 Activity Transition Detector

The algorithm for detecting human activity transitions in office-like environments has been shown to be effective and resistent to noise. This algorithm is an important step because it reliably detects activity transitions and is highly accurate at identifying when an activity transition does not take place. With only minor modifications, this algorithm could be implemented as a real-time detector in a context-aware system and allow that system to interrupt users at times more appropriate than those randomly selected. For example, this algorithm could be implemented as a CAES `Interaction` class.

The activity transition detection algorithm takes a novel approach to detecting the human behaviors of interest: activity transitions. By leveraging Borchardt's Transition Space representation [6], the need to identify particular activities disappears. This approach is simpler because it focuses the designer and the system at the most important results of the behavior to be identified instead of at the behavior itself.

Future work on this algorithm could follow one of two paths. First, its performance could be improved by identifying the activity transitions most likely to be misclassified. Second, it could be used as the basis to study when context-aware computer systems should interrupt people. In particular, future work should focus on validating the theory that interrupting users at activity transitions is better than interrupting them at random times.

Appendix A

CAES User Interface

This section describes the Context-Aware Experience Sampling Tool user interface in greater detail and discusses some of the design changes that were made in response to user feedback.

A.1 `Default` Screen

The `Default` screen has largely remained unchanged since the original design. This screen only allows the user to mute the survey for up to two hours. This functionality was required so that users could reasonably be expected to use CAES in their daily lives. For example, when going to a movie, attending a meeting, or studying at a library users must be able to prevent CAES from beeping at them. Figure A-1 shows all the possible views of the `Default` screen.

A.2 `Question` Screens

The CAES question screen underwent one major set of revisions. When first tested, subjects had a difficulty understanding the arrows at the bottom of questions spanning multiple screens due to a large number of possible answers. Subjects would often select the best answer from the first screen without realizing that there were more possible answers. Also, subjects who did click on the arrows often thought they were

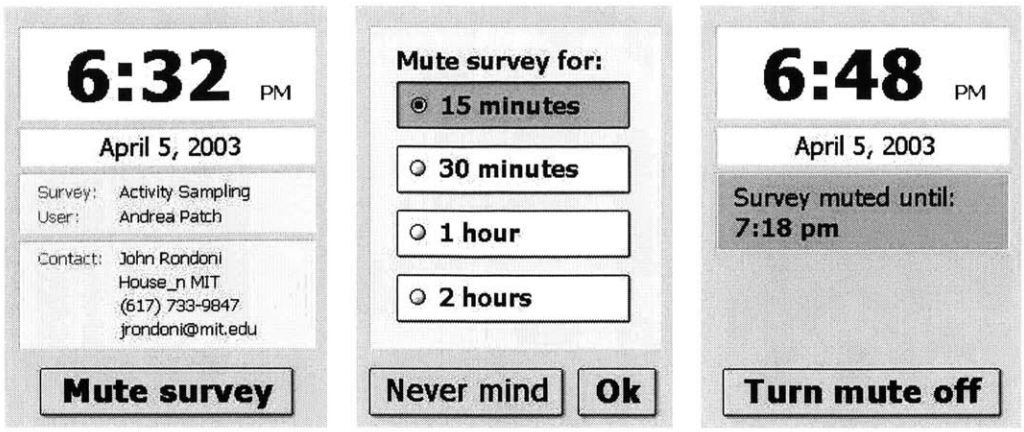

Figure A-1: the `Default` screens. The standard screen is visible to the user when the survey is not muted (left). By pressing the *Mute survey* button users can access the mute screen that allows them to select a time period to mute CAES (center). When CAES is muted the user is notified of the time when the mute ends and is provided with a button to end the mute on demand (right).

answering another question instead of viewing more answers to the same question.

To address these issues, directions were added to the bottom of the question screen. When the user can view more answers to the same question the text *More questions, touch arrow* is displayed. When the user has viewed all possible answers but has not selected one, the text *Select answer* is displayed. Once the user has selected an answer and viewed all the possible answers to a question, an *Ok* button replaces the directions so they can complete the question. Figure A-2 illustrates these modifications.

There are some cases where the researcher designing the survey cannot ensure that the questions asked of subjects will be applicable or that an appropriate answer will be available. To deal with such scenarios, researchers are provided with two options. First, they have the ability to allow subjects to select *Not now* instead of answering a question. The fact that the subject skipped a question is recorded by CAES. Second, researchers can enable subjects to answer questions using an audio or photo note. The photo note requires that the Pocket PC on which CAES is running has a camera installed. When the subject selects audio or photo note they are presented with a series of help screens that graphically and textually describe how they should use the

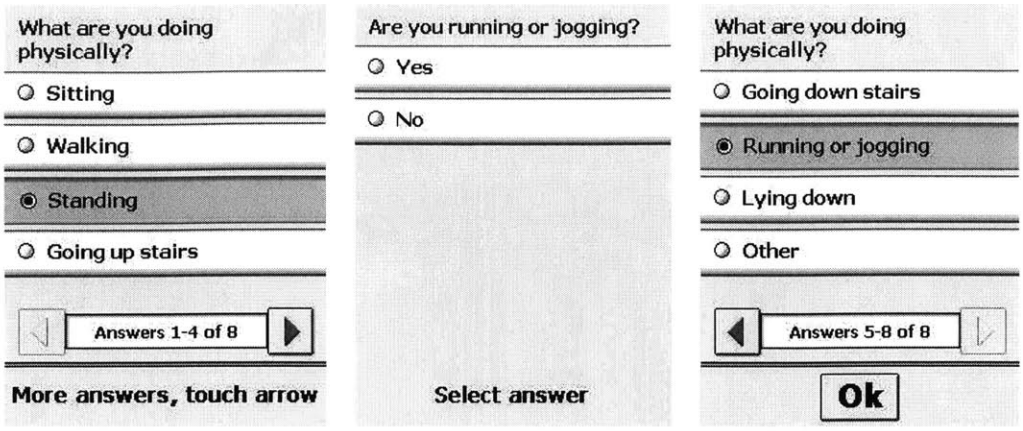

Figure A-2: the `Question` screen messages. When the subject has not viewed all the answers, they are directed to use the arrow to do so (left). When they have viewed all the possible answers or if there is only one screen of answers, the subject is directed to select an answer (center). Once the subject has viewed all the answers and selected one they are allowed to complete the question by pressing the *Ok* button (right).

sensor. Once the subject has taken a picture or left a message, their attached note is visible on the `Question` screen and they are allowed to select another answer if desired. Figure A-3 illustrates this functionality.

All question series are started and ended by a message screen. Before the start of a question series a *Touch screen to begin* message is displayed while CAES tries to get the subjects attention by beeping, unless it is muted. After the user touches the screen the beeping stops and the first question is displayed. At the end of a question series a *Thank you!* message is briefly displayed before returning the subject to the `Default` screen. Figure A-4 shows these message screens.

A.3 `Device` Screens

The `Device` screen is the simplest CAES screen. This screen presents the user with two simple text message while they are interacting with sensors and has no active elements. The researcher specifies the text displayed to the user while they are interacting with the sensor and when they have completed interacting with the sensor in the `QuestionDataFile`. The researcher can choose to provide a primary message,

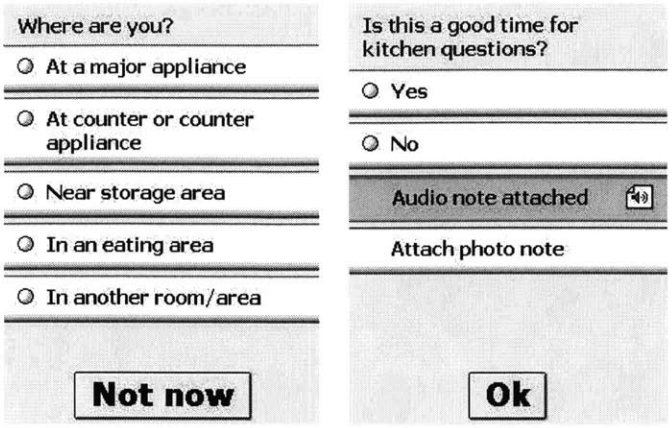

Figure A-3: the `Question` screen alternate answers. Researchers may allows subjects to opt-out of questions by selecting *Not now* when a question is first displayed (left). If their Pocket PC has the appropriate hardware, researchers may also allow subjects to answers questions with a photo or audio note (right). When selecting to answer a question with a device, subjects may also select from the list of researcher defined answers. Once a photo or audio note is attached it is visible to the subject and they are allowed to finish the question without providing any additionally response.

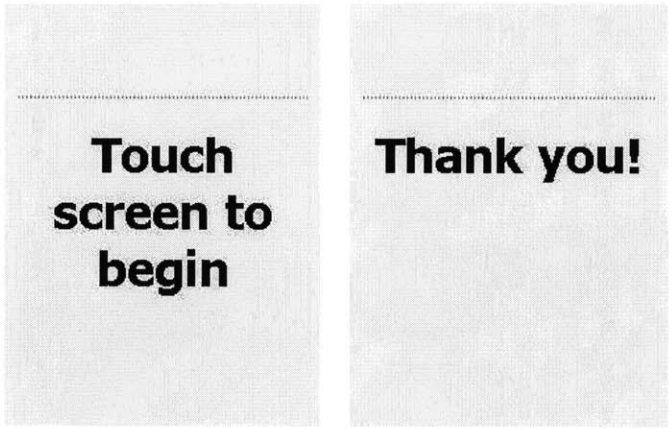

Figure A-4: Before the start of a question series the subject is presented with a simple screen while CAES beeps to get their attention (left). Once all questions in a series have been answered, a brief message is presented to the subject before they are returned to the `Default` screen (right).

which is displayed at the top of the screen in a large font, and a secondary message, which is displayed lower on the screen in a smaller font. The researcher may also choose not to provide any messages; however, this is likely to confuse subjects. Figure A-5 shows the recommended device screen messages for the microphone.

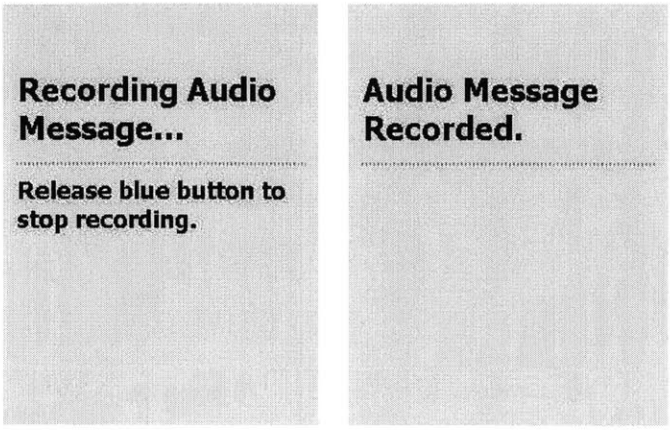

Figure A-5: When the user first activates the microphone they are displayed a message indicating that they are recording a message and with directions on how to stop recording (left). After they have stopped recording, the second `Device` screen messages are briefly displayed to confirm the successful recording (right).

For devices that have `Help` screens, the `Device` screen is displayed after the start `Help` screens and before the end `Help` screens (see Section A.4).

A.4 `Help` Screen

The `Help` screens are a very flexible tool for researchers. The primary purpose of this screen is to display a series of graphical and textual directions to subjects so they are able to operate sensors. For flexibility the locations of the bitmaps to be displayed are specified in the `QuestionDataFile`. This allows researchers to modify the existing `Help` screen and create new ones for special purposes. For example, `Help` screens could be created to prompt users to record data on a special purpose paper diary, such as a map.

There are two primary types of `Help` screens. Those displayed with a sensor

is triggered by CAES and those displayed when a user is attaching an audio or photo note to a question (see Section A.2). The `Help` screens displayed before a user activates a device to attach a note to a question allows the user to cancel the device interaction.

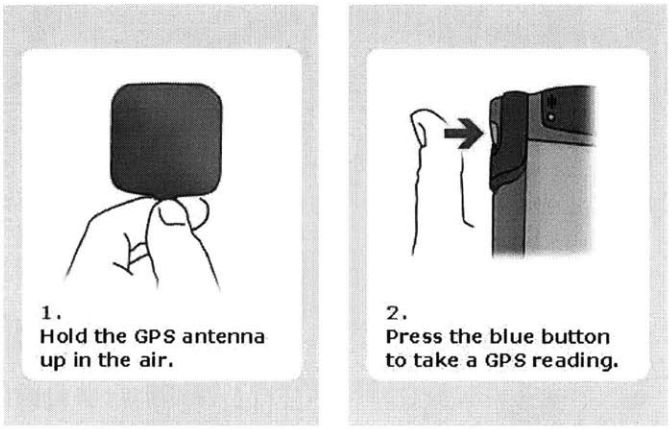

Figure A-6: These screens are displayed to subjects when a GPS reading is triggered by CAES. These `Help` screens are switched on the Pocket PC screen every couple seconds until the user activates the sensor to take a reading. Once activated the GPS `Device` screen is displayed

Figure A-7: These `Help` screens are rotated on the screen every couple seconds when the subject selects the *Attach audio note* option on the `Question` screen. These screens are displayed until the subject activates the microphone or they select *Never mind*. If the subject activates the microphone the appropriate `Device` screen is displayed. If the subject selects *Never mind* they are returned to the `Question` screen without attaching an audio note.

Appendix B

CAES Implementation Details

This Appendix discusses the most important aspects of the CAES implementation for programmers attempting to extend or modify CAES. Programmers only attempting to add devices or sensor to CAES should see Sections D and E.

B.1 Threading

CAES is a multi-threaded application. As a result, care must be taken when adding or modifying CAES code. Programmers must make sure they understand what data is shared between threads and take all necessary precautions to protect such data. There is not a significant overhead on Pocket PC for synchronizing data access. *Do not take shortcuts by avoiding proper thread-safe programming practices.*

There are at least two CAES threads at all times. The core scheduling system runs on the primary thread and the user interface runs on its own thread to ensure responsiveness.

The current user interface implementation introduces four additional threads. The user interface is implemented in four `CDialog` classes: `CQuestionDlg`, `CDeviceDlg`, `CHelpDlg`, and `CStartDlg`. Each of these classes is runs on their own thread. This allows the main user interface thread to preform background work such as preparing the next question while the user is answering the current one. When a dialog is not visible to the user its thread is generally blocking.

An unknown additional number of threads can be introduced by researchers' `QuestionDataFile` settings. Each sensor that is set by the `QDF` to collect data continuously spawns its own thread to do this. This has potential performance implications for CAES studies useing a large number of sensors that generate significant amounts of data. The upper limits on performance with large numbers of continuously sampled sensors has not been thoroughly tested.

B.2 Sensor and Interaction Instantiation

To simplify CAES and to allow for the integration of large numbers of sensor and interactions, instances of `Sensor` and `Interaction` classes are not created unless the researcher specifies it in the `QDF`. A little known or documented MFC implementation detail is used to convert the string provided by the researcher into a `CRuntimeClass` that can be used to create an instance of the `Sensor` or `Interaction`.

MFC keeps a global linked list of `CRuntimeClasses` for each Class in the application that is serializable. This is done so MFC can actively serialize to and from files. CAES reads the sensors and interactions specified by the researcher and then uses this global list to get `CRuntimeClasses` to instantiate. This is done by comparing the string from the `QDF` to the names of `CRuntimeClass` in the global list.

There are two significant implications of this approach. First CAES uses an MFC implementation detail; functionality not documented by Microsoft. As a result, this approach is not guaranteed to work in future MFC versions. If you upgrade MFC and find CAES breaks check this first. It would not be overly difficult to replace this approach with a global registry of all `Sensor` and `Interaction` classes.

The second implication of this approach is for `Sensor` and `Interaction` classes. The implementation of these classes must ensure that they are serializable so their `CRuntimeClass` is included in the global MFC list. This is easily done with two MFC macros (see Section D.1.2). See MFC documentation or the `DefaultInteraction` and `DefaultSensor` classes for more details.

B.3 Pocket PC Timers

Pocket PC implements system timers in a similar to fashion to desktop versions of Microsoft Windows. However, they do not work for nearly all purposes in CAES. This is because Pocket PCs are turned off as soon as the user presses the prominently placed power button. When this happens, the internal processor is stopped as well as the screen being powered down. As a result, if the system is turned off timers based on the processor clock will not be accurate.

Pocket PC provides the Notification API to address this problem. Notifications can act like timers that cause the system to be woken. However these are difficult to use and manage. To address this problem the `CTimers` class was written. Callback timers crated with the global `CTimers` class maintained by the core scheduling system are implemented with the notification API and so are assured to occur at the appropriate times. Programmers are strongly urged to user this method to add timing functionality to new CAES classes instead of creating a new mechanism.

Appendix C

Activity Detector Study Data

This Appendix presents some typical heart rate and planar accelerometer data from a few subjects. The heart rate data presented has been cleaned according the process described in Section 4.2.6.

In the following data is from nine different subject over different time periods. Effort was take to make sure this data is generally representative of the type collected.

Figure C-1 shows the data collected from three subjects over the entire business they were studied.

Figure C-2 shows the data from three subjects over the entire period of about one hour.

Figure C-3 shows the data from three subjects over the entire period of about ten minutes.

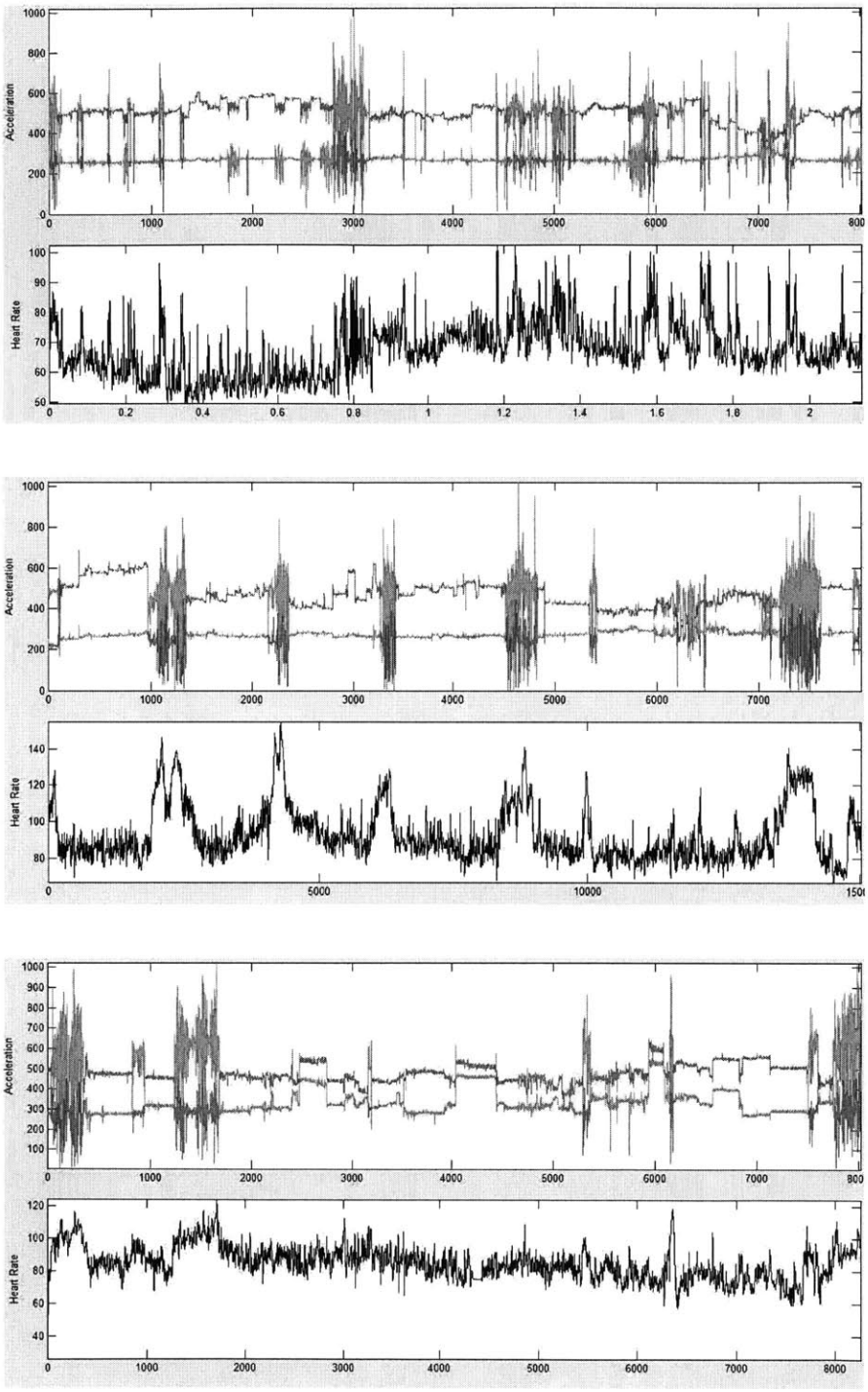

Figure C-1: These graphs show the planar accelerometer and heart rate data collected from three different subject for the duration of the study. The heart rate data has been cleaned. This is the data that is segmented, compared, and used for machine learning

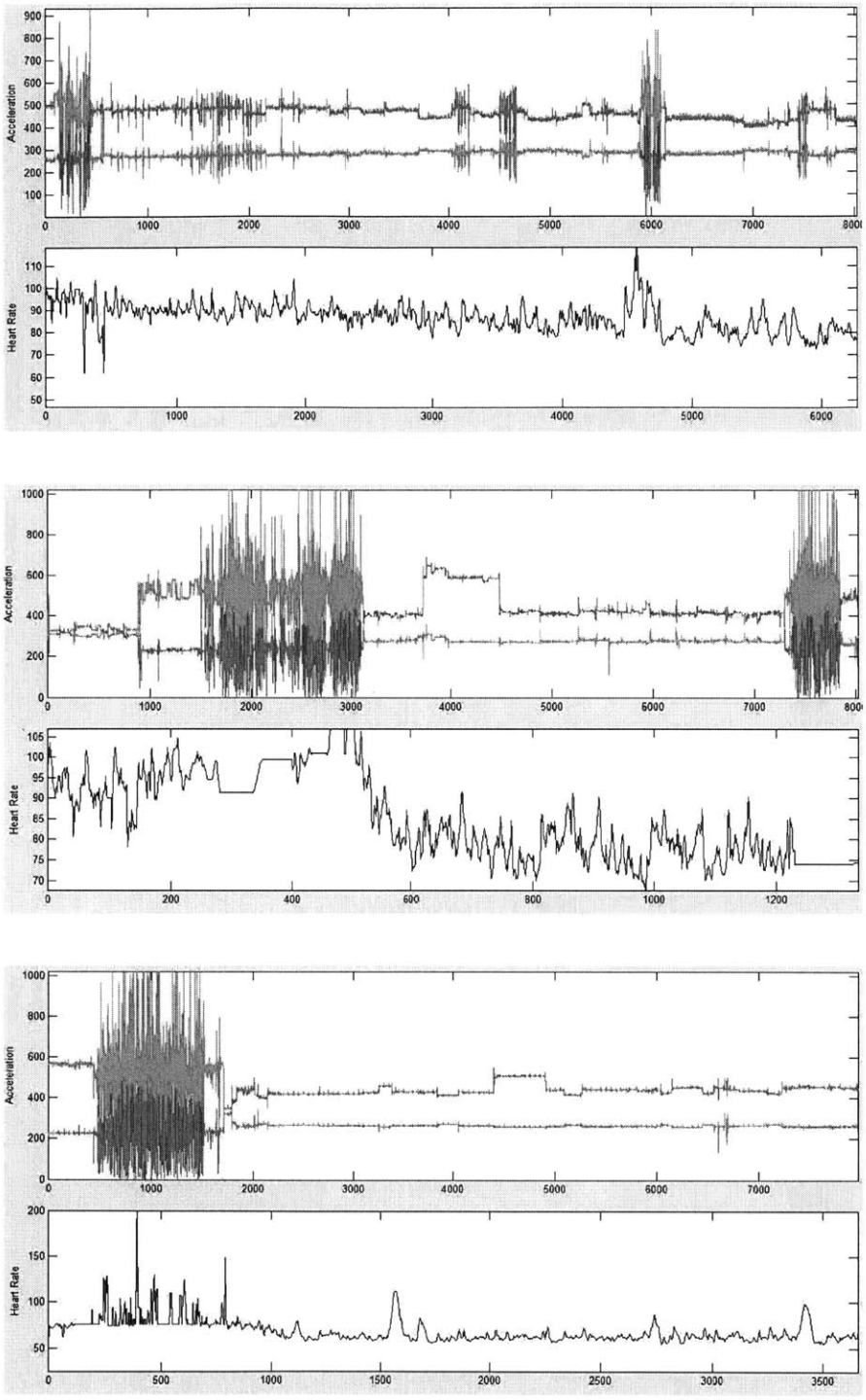

Figure C-2: These graphs show the planar accelerometer and heart rate data collected from three different subject for one hour. The heart rate data has been cleaned. This is the data that is segmented, compared, and used for machine learning

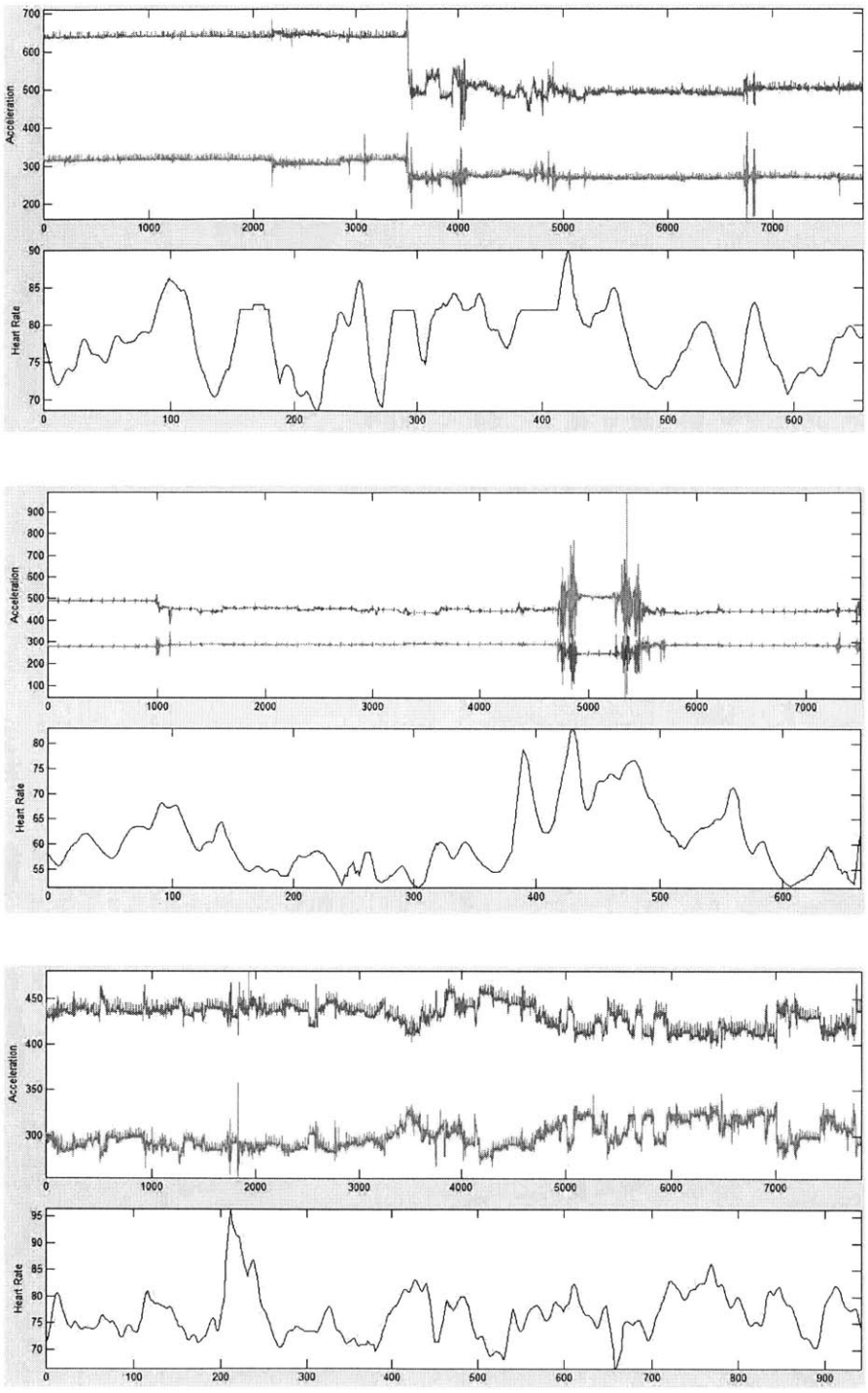

Figure C-3: These graphs show the planar accelerometer and heart rate data collected from three different subject for ten minutes. The heart rate data has been cleaned. This is the data that is segmented, compared, and used for machine learning

Appendix D

Adding Interactions

This Appendix presents the important points of developing a new `Interaction` class.

D.1 Defining the Class

All new interactions are implemented in `Interaction` subclasses. There are three important steps to defining a new interaction: choosing the class name, making the class serializable, and extending `Interaction`.

D.1.1 Naming Considerations

The names of `Interaction` classes are more important than most class names in CAES and other programs. CAES translates the interactions specified by the researcher directly into a `CRuntimeClass` that instantiated. As a result, researcher who will want to use the new interaction will have to be familiar with its name. `Interaction` class names therefore must be not be accurately describe the functionality of the interaction.

D.1.2 Making a Serializable Class

For CAES to retrieve a `CRuntimeClass` to instantiate a version of the new `Interaction` class it must be a serializable class. The has to do with the inner workings of

MFC; however it is easy to make your class serializable. For an example look at the `DefaultInteraction` class. Also the process of making a class serializable is well documented by Microsoft.

To make the new class serializable add the macro call `DECLARE_SERIAL(`*`CNewClassInteraction`*`)` to a private area of the class definition. Then add the macro call `IMPLEMENT_SERIAL(`*`CNewClassInte`*`CObject, 1)` to the implementation file for the new class. Replace *CNewClassInteraction* with the name of the new `Interaction` class.

D.1.3 Extending `Interaction`

CAES is a multi-threaded application. To ease the burden on the programmer writing only a new `Interaction` class all thread protection and core scheduling system functionality is implemented in the abstract `Interaction` class. As a result, this should not be a concern.

The multi-threaded character of CAES is important to remember for two reasons. First, there is a lot of work to be done so make sure to be as efficient as possible in the implementation of new interactions. If an interaction will be slow by its nature warn researchers so they know what to expect when using it. Second, it is extremely important that the `RunInteraction()` guidelines are followed.

D.2 Implementing `RunInteraction()`

The interaction function will be called regularly by the core scheduling thread. This will allow the `Interaction` class to perform the necessary processing to detect events of interest and notify the core scheduling system.

D.2.1 Functions Provided

There are seven functions provided by the `Interaction` class that will be used in `RunInteraction()`. The first two functions, `UserDataAvailable()` and `SensorDataAvailable()` return true when data is available to be processed. When data is available, the next

two functions `GetNextUserAnswer()` and `GetNextSensorData()` should be used to retrieve the next user response or chunk of sensor data. Once `RunInteraction()` is done processing the user response or sensor data, `Handled(CSensorData* data)` or `Handled(CUserAnswer* answer)` should be called so the core scheduling system can store the data and de-allocate its memory. The final function call is only used when an interesting event is noticed in the user responses and sensor data. In that case, `SendSignal(CA_SIGNAL signal)` should be called with the appropriate `EventTag`.

D.2.2 Example Implementation

```
void CDefaultInteraction::RunInteraction()
{
    CSensorData* data;
    CUserAnswer* answer;
    while(UserDataAvailable())
    {
        answer = GetNextUserAnswer();
        // Do Processing Here
        SendSignal(CA_DEFAULT);
        Handled(answer);
    }
    while(SensorDataAvailable())
    {
        data = GetNextSensorData();
        // Do Processing Here
        SendSignal(CA_DEFAULT);
        Handled(data);
    }
}
```

D.3 Adding EventTags

Once a new `Interaction` class has been implemented, the `EventTags` it will use to communicate the occurrence of interesting events the core scheduling system must be defined.

D.3.1 Creating CA_SIGNAL

The `CA_SIGNAL` type is defined in `types.h`. The first step in defining an event tag is to add it to `CA_SIGNAL`.

D.3.2 Modifying CEngine::ReccuranceSignalStrings

Once the new `CA_SIGNAL` is defined add, the text string researchers should use to refer to the new `EventTag` to the `CEngine::ReccuranceSignalStrings` array in the position corresponding to that of the new `CA_SIGNAL`.

At this point the new event tag has been added. The `CEngine::ReccuranceSignalStrings` array is only used for translating the `QuestionDataFile` text into the internal `CA_SIGNAL` representation. Once the new `EventTag` has been added there is no reason to ever refer to its string representation in the `CEngine::ReccuranceSignalStrings` array.

Appendix E

Adding Sensors

The process of adding new sensors to CAES is quite painless. All of the work necessary for thread protection and integrating new `Sensor` classes into the core scheduling system has already been taken care of. Programmers adding new sensors will only have to write code specific to the sensor being added.

E.1 Defining the Class

The first set in adding a new sensor to CAES is to create a new sensor class by extending `Sensor`.

E.1.1 Naming Considerations

The name of a Sensor class is important. The class name is how researchers specify the use of a sensor in the `QuestionDataFile`. The class name should be descriptive of the sensor being implemented. If specific manufacturer names and model numbers can be included in the class name much confusion will be avoided.

E.1.2 Making a Serializable Class

For CAES to successfully convert the class name provided in the `QDF` to an instance of the class it must be serializable. This process is explained in greater dail in Sec-

tion D.1.2.

E.2 Implementation

All new sensor classes must implement the six abstract functions they inherit from the `Sensor class`. However, programmer do not need to worry about thread protection or core scheduling system interfacing, all this has already been taken care of in the `Sensor` class

E.2.1 Abstract `Sensor` Functions

All sensors need to implement the six abstract `Sensor` functions: `SensorInitialize()`, `SensorSampleStart()`, `SensorSampleStop()`, `SensorStoreData(CSensorData *sd)`, `SensorSample()`, and `SensorContinuousSample()`.

The `SensorInitialize()` function is where all device initialization should be performed. If a `DataSource` class is being used it should be initialized here. If a file must be opened to store sensor data to it should be opened here and closed when the class is destroyed.

The `SensorSampleStart()` function is called when the sensor is started and stopped at different times. For example the microphone sensor is started when the subject depresses a button.

The `SensorSampleStop()` function is called following `SensorSampleStart()` when the sensor sample should be stopped. Sensors unlike microphones that always take instantaneous samples can implement all the sample functionality here because this is guaranteed to be called following `SensorSampleStart()`.

The `SensorStoreData(CSensorData *sd)` function is called after the data collected by a sensor has been acted on by all active `Interaction` classes. The data can be stored or the data can be discarded.

The `SensorSample()` function should perform a quick and discrete sample of the sensor. This could be considered a successive call to `SensorSampleStart()` and `SensorSampleStop()`.

The `SensorContinuousSample()` function should act similarly to `SensorSample()`, however if the sensor is continuously being sampled this function will be called by the sampling thread in case special measures should be taken.

E.2.2 Example Implementations

```
bool CDefaultSensor::SensorInitialize()
{
    return true; // Do Initialization Here
}

bool CDefaultSensor::SensorSampleStart()
{
    return true; // Start Sample
}

CSensorData* CDefaultSensor::SensorSampleStop()
{
    // Finish Sample
    return new CSensorData(this, CTime::GetCurrentTime(), /*Data Pointer*/NULL);
}

bool CDefaultSensor::SensorStoreData(CSensorData *sd)
{
    // Store Data To Appropriate Directory / Repository
    return true; // return false if store fails
}

CSensorData* CDefaultSensor::SensorSample()
{
    // Do Complete Sample
```

```
    SensorSampleStart();
    return SensorSampleStop();
}

CSensorData* CDefaultSensor::SensorContinuousSample()
{
    return SensorSample();
}
```

Bibliography

[1] Weka machine learning project. http://www.cs.waikato.ac.nz/~ml/index.html.

[2] G.D. Abowd and E.D. Mynatt. Charting past, present, and future research in ubiquitous computing. *ACM Transactions on Computer-Human Interaction*, 7(1):29–58, 2000.

[3] K. Aminian, P. Robert, E. Jequier, and Y. Schutz. Estimation of speed and incline of walking using neural network. *IEEE Transactions on Instrumentation and Measurement*, 44(3):743–746, 1995.

[4] I. Anacona. Context-aware experience sampling tool user interface, November 19 2002.

[5] L.F Barrett and D.J. Barrett. The experience sampling program. http://www2.bc.edu/~barretli/esp/.

[6] G.C. Borchardt. Causal reconstruction. Technical Report A.I. Memo No. 1403, Massachusetts Institute of Technology, February 1993.

[7] C. V. Bouten, K. T. Koekkoek, M. Verduin, R. Kodde, J. D. Janssen, Division of Computational University of Technology, and Eindhoven The Netherlands carlijn wfw tue nl Experimental Mechanics. A triaxial accelerometer and portable data processing unit for the assessment of daily physical activity. *IEEE transactions on bio-medical engineering.*, 44(3):136–47, 1997.

[8] C.V.C. Bouten, A.A.H.J. Sauren, M. Verduin, and J.D. Janssen. Effects of placement and orientation of body-fixed accelerometers on the assessment of energy

expenditure during walking. *Medical & Biological Engineering & Computing*, 35(1):50–56, 1997.

[9] Kathy Burns, May 6th 2003.

[10] J. B. Bussmann, J. H. Tulen, E. C. van Herel, and H. J. Stam. Quantification of physical activities by means of ambulatory accelerometry: A validation study. *Psychophysiology*, 35(5):488–96, 1998.

[11] CAES. Context-aware experience sampling website, 2003. http://caes.sourceforge.net.

[12] O. Cakmakci, J. Coutaz, K. Van Laerhoven, and H. Gellersen. Context awareness in systems with limited resources. In *Artificial Intelligence in Mobile Systems (AIMS 02)*, 2002.

[13] M. Csikszentmihalyi and R. Larson. Validity and reliability of the experience-sampling method. *The Journal of Nervous and Mental Disease*, 175(9):526–36, 1987.

[14] R.W. DeVaul and S. Dunn. Real-time motion classification for wearable computing applications. Technical report, MIT Media Laboratory, 2001.

[15] P. Dragicevic and S. Huot. Spiraclock: A continuous and non-instrusive display for upcoming events. In *CHI 2002*, pages 604–5, Minneapolis, Minnesota USA, 2002.

[16] Polar Electro. website. http://www.polar.fi.

[17] F. Foerster, M. Smeja, and J. Fahrenberg. Detection of posture and motion by accelerometry: a validation in ambulatory monitoring. *Computers in Human Behavior*, 15:571–583, 1999.

[18] H. A. Hayden-Wade, K. J. Coleman, J. F. Sallis, and C. Armstrong. Validation of the telephone and in-person interview versions of the 7-day par. *Med Sci Sports Exerc*, 35(5):801–9, 2003.

[19] D. Hendelman, K. Miller, C. Baggett, E. Debold, and P. Freedson. Validity of accelerometry for the assessment of moderate intensity physical activity in the field. *Medicine & Science in Sports & Exercise*, 32(9 Suppl):S442–9, 2000.

[20] R. Herren, A. Sparti, K. Aminian, and Y. Schutz. The prediction of speed and incline in outdoor running in humans using accelerometry. *Medicine & Science in Sports & Exercise*, pages 1053–1059, 1998.

[21] S.E. Hudson, J. Fogarty, C.G. Atkeson, D. Avrahami, J. Forlizzi, S. Kiesler, J.C. Lee, and J. Yang. Predicting human interruptability with sensors: A wizard of oz feasibility study. In *Proceedings of the Conference on Human Factors and Computing*. ACM Press, 2003.

[22] Polar Electro Inc. Adding heart to your technology: New smart coded reciever, 2003.

[23] S.S. Intille, J. Rondoni, C. Kukla, I. Anacona, and L. Bao. A context-aware experience sampling tool. In *Proceedings of the Conference on Human Factors and Computing Systems: Extended Abstracts*. ACM Press, 2003.

[24] invivodata. website. http://www.invivodata.com.

[25] Nicky Kern, Bernt Schiele, Holger Junker, and Paul Lukowicz. Wearable sensing to annotate meeting recordings. In *International Symposium on Wearable Computers (ISWC)*. IEEE Press, 2002.

[26] A. Madabhushi and J.K. Aggarwal. A bayesian approach to human activity recognition. In *Proceedings of the Second IEEE Workshop on Visual Surveillance*. 1998.

[27] Vijay Mahajan and Jerry Wind. New product models: Practice, shortcomings and desired improvements. *The Journal of Product Innovation Management*, 9(2):128–139, 1992.

[28] C.E. McCarthy and M.E. Pollack. A plan-based personalized cognitive orthotic. In *6th International Conference on AI Planning and Scheduling*, Toulouse, France, 2002.

[29] T.L. McKenzie. Use of direct observation to assess physical activity. In J.G. Welk, editor, *Physical Activity Assessments for Health Related Research*, pages 179–195. Human Kinetics, 2002.

[30] MIT. Changing places / house_*n*: Mit home of the future consortium, 2002. http://architecture.mit.edu/house_*n*.

[31] E. Munguia Tapia, S.S. Intille, and K. Larson. Activity recognition in the home setting using simple and ubiquitous sensors. House_*n* technical report, MIT, 2003.

[32] N. Oliver, E. Horvitz, and A. Garg. Layered representation for human activity recognition. In *Fourth IEEE International Conference on Multimodal Interfaces*, page 3, Pittsburgh, Pennsylvania, 2002.

[33] M. E. Pollack, S. Engberg, J. T. Matthews, S. Thrun, L. Brown, D. Colbry, C. Orosz, B. Peintner, S. Ramakrishnan, J. Dunbar-Jacob, C. McCarthy, M. Montemerlo, J. Pineau, and N. Roy. Pearl: A mobile robotic assistant for the elderly. In *AAAI Workshop on Automation as Eldercare*, 2002.

[34] C. Randell and H. Muller. Context awareness by analysing accelerometer data. In Blair MacIntyre and Bob Iannucci, editors, *The Fourth International Symposium on Wearable Computers, IEEE Computer Society*, pages 175–176. IEEE Press, 2000.

[35] J.P. Robinson. The validity and reliability of diaries versus alternative time use measures. In F.T. Juster and F.P. Stafford, editors, *Time Goods and Well-Being*, pages 33–62. Ann Arbor, MI, 1999.

[36] B.G. Silverman. Computer reminders and alerts. *IEEE Computer*, 30(1):42–49, 1997.

[37] A.A. Stone and S. Shiffman. Ecological momentary assessment (ema) in behavioral medicine. *Annals of Behavioral Medicine*, 16(3):199–202, 1994.

[38] A.A. Stone, S. Shiffman, J.E. Schwartz, J.E. Broderick, and M.R. Hufford. Patient non-compliance with paper diaries. *British Medical Journal*, 324(7347):1193–4, 2002.

[39] K. Van Laerhoven and O. Cakmakci. What shall we teach our pants? In *The Fourth International Symposium on Wearable Computers*, pages 77–83. IEEE Press, 2000.