6-1-2001

SmartReminder: A Case Study on Context-Sensitive Applications

Arun Mathias
Dartmouth College

SmartReminder: A Case Study on Context-Sensitive Applications

Arun Mathias
Arun.Mathias@alum.dartmouth.org
Senior Honors Thesis
Dartmouth College Computer Science
Advisor: David Kotz

Technical Report TR2001-392

1 June, 2001

Abstract

Designing context-sensitive applications is challenging. We design and implement SmartReminder to explore designing context-sensitive applications and to demonstrate how the SOLAR system can be used in developing such applications. SmartReminder is an application that reminds the user based on contextual information. Current appointment-reminder applications remind the user about their appointments at an arbitrarily specified time. For instance, they might remind the user ten minutes before each appointment. SmartReminder, on the other hand, uses contextual information, like location, to better estimate the appropriate reminder time for each appointment. It reminds the user based on where they are, where they need to be, and how long it will take them to get there. This paper presents SmartReminder as an illustration of how context-sensitive applications can be designed using the SOLAR system for dissemination of contextual information.

1 Introduction

A *context-sensitive application* is one that is aware of its context and acts or adapts accordingly. The context is the environment in which the application is running. It is determined by many types of context information. With the advent of wireless networking, allowing for easy mobility, location is an increasingly important part of an application's context. Other aspects of the context include weather, lighting, noise level, network connectivity and traffic conditions. The applications need sensors to detect their context information and remain aware of their environments.

This project was supported by the Cisco University Research Program, by Microsoft Research, and by DoD DURIP contract number F49620-00-1-0212.

This thesis explores the design of context-sensitive applications. It starts by giving a broad introduction into issues that one must consider when designing such applications. Next, it shows how *SmartReminder*, the application we developed, is one such context-sensitive application. The rest of this paper focuses on SmartReminder: its design, its operation, and how it deals with the various issues encountered by context-sensitive applications. The goal is to demonstrate a method for designing context-sensitive applications, and SmartReminder is a sample application.

1.1 Context-Sensitive Applications

Context-sensitive applications are aware of their surroundings. They adapt to changes in context such as location of use, collection of nearby people, hosts on same subnet, schedules for the day, and so forth. Context-sensitive applications can be further divided into the following categories [5]:

Awareness of proximate resources: Applications that detect resources, places or people of interest to the user fall under this category. For example, they might detect the printers or video cameras that are close by, or display a list of nearby restaurants.

Automatic Contextual Reconfiguration: This category includes applications that reconfigure devices based on the current context. For example, a device that is sensitive to the lighting in its environment can adjust the brightness of the screen. Similarly a device might keep its disk spinning for longer if it has a power connection.

Contextual Information and Commands: Such applications adapt so that when a request is received, the information they provide is different based on location. Consider a web browser whose home website differs based on location. If you are in a dining area, and open a browser it might show you the menu, while if you are in the music center it might show you the list of performances.

Context-triggered Actions: These applications perform actions, such as notifying the user, based on a predefined set of rules about contexts. They are different from the previous

category in that their actions are automatic and they are not dependent on the user requesting information. SmartReminder is one such application.

Designing applications that are aware of their contexts is challenging. The challenge lies in the complexity of capturing, representing and processing contextual data. An application should have an efficient way to receive contextual information and adapt quickly to it.

2 Challenges of Context-Sensitive Applications

While designing a context-sensitive application as described above, one must be aware of the following issues and incorporate them into the design:

1. These applications are heavily dependent on sensors. But, most sensors send redundant information. For example, a location sensor will constantly send updates about the location of the device. Since for most contexts, the application is only interested in changes, it should be notified only when such a change occurs. If the application does indeed need all information, it should be able to receive the raw stream of sensor output.

2. The application depends on several types or sources of context information. Some of the context information might have conflicting implications. Applications must be designed to handle such a possibility smoothly. To resolve the conflict, certain context information must take precedence over others, or all information must be considered before deriving an interpretation of the information.

3. These applications are useful if they work on devices that can be carried around by their users. These devices are often less powerful than conventional desktops. The application design should take this into account and minimize the size of the application running on the mobile device.

4. To make the application that runs on the device thin, it might have to use the network extensively. Since the application uses the wireless network, however, the user might venture into areas where the signal is weak or non-existent. The applications should gracefully deal

with the possibility of disconnection from network resources. It should also immediately realize when the network connection has been restored and return to performing like normal.

5. Similarly, the application might lose contact with the sensors that detect contextual information. Again, it should gracefully continue functioning as best it can until sensor information is available, at which time it should resume normal functioning.

6. Contextual information is the most important piece of information for these applications. Some of the information they receive, however, might not be totally accurate. For instance, if the location sensor places the device at a certain location, how certain is the application that the user is actually at that location? Applications should compensate for possible uncertainty in the contextual information they receive. There will also be information that is certain. Applications should know how to properly interpret the information they get.

7. Section 4 introduces location-sensing systems. These systems work either outdoors or indoors. Most context-sensitive applications require location information both when the user is outdoors and indoors. Therefore, applications might need to work with location information from two systems, one for outdoors and one for indoors. Integrating information from these two systems could be a complex task because the two systems might represent location differently [10].

3 SmartReminder

There are many reminder applications available that remind the user about meetings on his/her schedule, for example, by beeping or by popping up a window on the computer screen. A vast majority of them, however, do not have the ability to discern the appropriate time to notify the user of a meeting or event. A context-sensitive application that is aware of its environment should choose a reminder time appropriate to the context of the user. In particular, it should remind the user about a meeting based on his/her current location and how long it will take to get to the meeting venue. Such an application must be aware of several types of context information. These include:

Time: This is the most trivial piece context information that a reminder should use, and one that most, if not all, reminders currently use.

Location: Location is a powerful piece of data to consider when deciding the time and content of the reminder. Location sensors give the application the ability to answer questions like: Where is the user now? How far is the user from the place they have to be soon? Answering these two questions makes the application extremely useful. For example, if Charles has a meeting down the hall from his current location, he only needs a reminder a few moments before the meeting. But, if his meeting is across town, he should be reminded early enough so that, had he forgotten about the meeting, he would still make it on time. Alternatively, if the application determines that he is on his way to the meeting, it does not need to notify him, and hence can cancel or postpone the reminder.

User's schedule: A reminder application reminds the user of meetings on the schedule, so it is essential that the application be aware of any changes. If an appointment is added, deleted or modified, the reminders are updated accordingly. Since the application knows the locations of the meetings and the time it takes to get to them, it can alert the user ahead of time about meetings that could potentially be hard to make. It may also be possible to use the location of other users who are slated to attend a meeting. Using this information, the application could decide if another person is running late, and delay the reminder.

Proximate Users: The application can be aware of the location of other users. Using this information, the application has the ability to remind a user about something he/she needs to discuss when they are with a certain person. Often we find ourselves wanting to discuss something with someone, but forgetting when we are actually with the person.

Weather and Traffic: Weather is a major factor to consider when calculating the time needed to reach a destination. So too is traffic. Although weather and traffic can drastically affect travel time, they are difficult to quantify. We have not found a quantification, and save its integration for future work.

We designed *SmartReminder*, a prototype context-sensitive reminder application, using most of the types of context information listed above. In addition to adapting to different contextual information, SmartReminder must account for many of the common issues listed above.

4 Location-Sensing Systems

Location is clearly an important piece of information for most context-sensitive applications. In the case of SmartReminder, it is probably the most important. If we look at the list of contextual information the application receives, the main decision on when to remind is based on the user's distance from the destination. Some of the other types of context information are used to improve the estimate of best reminder time.

Commercial systems that detect location in an outdoor setting typically use Global Positioning System (GPS) or cellular phone networks. Systems that detect location indoors, however, are less commonly available. Designing a system to detect the location of a device indoors is complex for a number of reasons. These reasons include administration and management overhead, system scalability and the harsh nature of indoor wireless channels. Indoor environments often contain a substantial amount of metal and other such materials that distort the propagation of radio frequency signals, making it hard to detect location using such signals [8].

Since determining location is imperative for the proper functioning of SmartReminder, and since SmartReminder is geared toward the office professional who is largely indoors, we consider a few location-sensing systems that are designed to work indoors. We present three representative systems here.

4.1 The Active Badge System

The Active Badge system was developed at Olivetti Research Lab in the early 90's [6]. The office personnel wore badges that transmitted IR signals. Sensors around the building picked up these signals and reported them to a central server. An application could query the central server to

obtain the location of any person.

4.2 The Cricket Location-Sensing System

Cricket is a location sensing system developed at MIT [8]. It uses *beacons* to disseminate information about a geographic space to *listeners*. A beacon is a small device attached to some location and represents a certain space, such as an office or conference room. A listener is a small device that can be attached to the mobile device carried by the user. The listener detects messages from the beacon, and uses these messages to identify the space it is currently in. The boundaries between adjacent spaces can either be real — like a wall — or virtual, such as regions of a large lecture hall. The system has an API to allow applications on the mobile device to determine the device's location.

4.3 The Nibble Location System

Nibble is a location-sensing system under development at UCLA [9]. It works differently than the others in that it does not use special hardware to determine the location of the user carrying a wireless computer. Instead, it uses the signal quality from the access points in the wireless data network, as reported by the computer's wireless card. During a training phase, the system learns the signal quality to expect at various locations. It takes information about the location, a name for example, and incrementally builds a Bayesian network based on the signal quality at that location. This information can be used at a later stage to determine the most likely location based on the signal quality signature. This system also has an API that can be used by programs on mobile devices to determine location. Nibble returns a list of probable locations with the probability estimate of being at each of those locations.

There is an element of uncertainty involved in the location estimate that the mobile device receives from these systems. SmartReminder must account for this uncertainty when it makes its decision on the best time to remind. The design of the SmartReminder application incorporates the uncertainty, as described in the next section.

5 SmartReminder Application Design

Context-sensitive applications require large amounts of information about their surroundings. They must run in a framework that supports the dissemination of the context information collected by sensors. This framework should scale easily to support diverse applications running on dozens of mobile devices. It should also abstract the context-sensing layer from the application. The core application should not require modification to support a different sensing system. SmartReminder uses the Solar system [12], developed at Dartmouth, to enable it to properly receive and adapt to contextual information. There are other systems similar to Solar under development, which are discussed in Section 7.1. We believe, however, that these systems lack sufficient scalability and flexibility.

5.1 The Solar System

Solar represents contextual information as events. Sensors are referred to as *sources* and they publish a stream of events. An application subscribes to *event streams* — an event stream is a sequence of events — that it cares about and adapts to events it receives. Context-sensitive applications are generally not interested, however, in a raw stream of events from a sensor. Applications can filter, transform, merge or aggregate these event streams by defining objects that support such operations. These are called *operators*; the application programmer composes operators into a tree that transforms raw sensor events into the high-level events the application needs.

Solar implements events as objects. Events are of a certain class that defines their type. A publisher produces an event stream, and an event subscriber receives the event stream and handles each event accordingly. Each operator receives one or more event streams, and produces a new event stream. Operators can subscribe to sources or other operators, but they need to form an acyclic graph of event flow, called an operator graph. An application extends the existing Solar event graph, as needed, to produce the desired event stream, and subscribes to that event stream.

The Solar approach allows applications to receive events that meet their needs. It also allows for

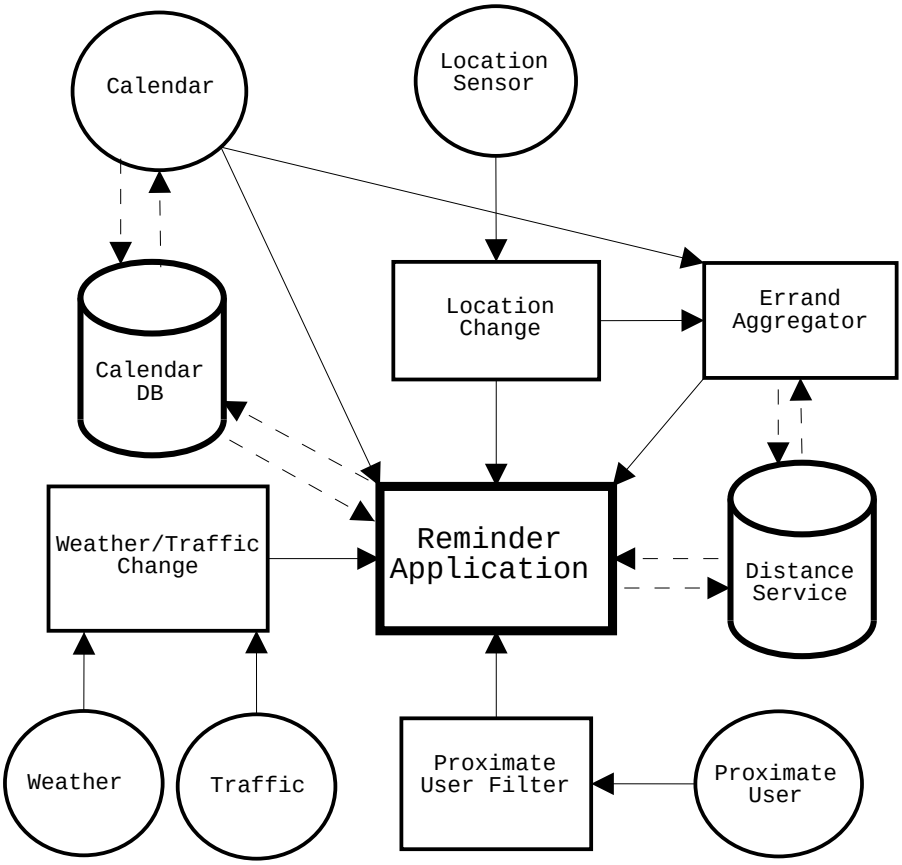

Figure 1: SmartReminder Design

an object-oriented design, which means that operators can be reused, and the context-sensing layer can be abstracted from the application. We exploit these properties in the design of SmartReminder.

5.2 SmartReminder Design using Solar

The SmartReminder design is illustrated in Figure 1. The circular objects represent sources and the rectangular objects represent operators. The diagram is one possible design and makes assumptions about the working of the sensing systems. This section is dedicated to explaining this figure and showing how the application works using this design. It also demonstrates how the design can be modified to support other sensing systems. Subsequent subsections demonstrate how this design solves the complexities we expected to encounter with this context-sensitive application.

5.2.1 Reminder Application Adapting to Changes in Context

Operators notify the application of changes in context information. The notification is in the form of an event (represented by solid arrows in the figure). These events are objects. They derive from a base *ReminderContextChangeEvent* type and have *updateReminder* methods that the application calls to update the reminder times. The application passes the schedule, current context and list of the reminder times to the method, and the method modifies the reminder times of the appropriate appointments. The events handle the task of modifying the reminders and, therefore, are specific to the reminder application. This approach minimizes the modifications necessary to the core application to support new types of context information. One might argue that the approach is not modular because we depend on the context-sensing layer to provide application-specific events. To support the use of non-application-specific sensing systems, we simply add an additional operator to take a generic event from the context-sensing layer, and publish a *ReminderContextChangeEvent* to the application. Thus, we minimize the modifications necessary to the core application to support new types of context information, yet maintaining the modular nature of context-sensing systems. The remainder of this section discusses the design at a higher level and is not concerned with this particular issue.

5.2.2 Schedule Information

A central database holds the schedule of each user. Several applications can modify a user's schedule because the database resides on an accessible server. SmartReminder reads this database when it starts up. To ensure that it always has a current version of the schedule, it must be aware of changes to the database. The schedule can be kept current either by SmartReminder continually polling the database for changes, or by events notifying SmartReminder of a change. The former approach is inefficient. Therefore, we choose the latter approach as shown in Figure 1. A *Calendar* source, which is an application that can be used to modify the schedule, both updates the database and notifies SmartReminder of the change. SmartReminder receives the notification and adapts accordingly. This design ensures SmartReminder always has an accurate schedule.

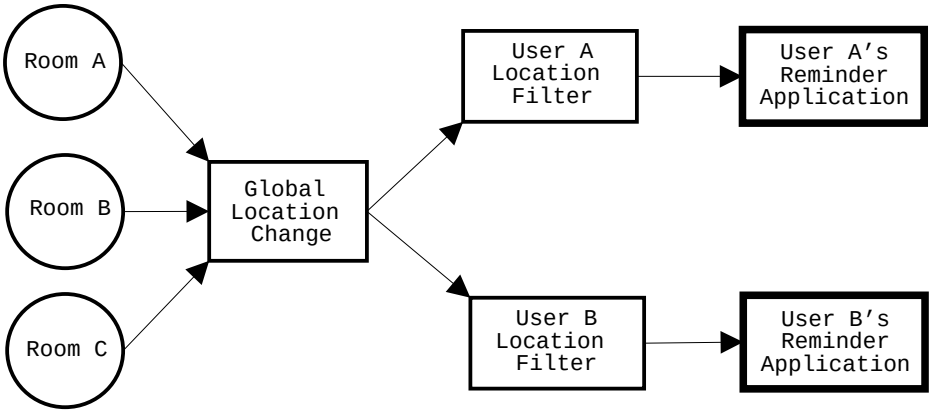

Figure 2: Alternate Location Support

5.2.3 Location Information

Location is an important form of context information in this application. The design for handling location information is somewhat dependent on the type of location-sensing system we use. The system could either be one in which the mobile device senses its own location, or one in which the sensor in a location senses devices in its space. The Cricket and Nibble sensing systems described in Section 4 use the former approach and Active Badge used the latter approach. Figure 1 assumes the former approach. The location sensor, a source on the mobile device itself, continually publishes information about the location of the device. The application is only interested in knowing about changes in location, so a *Location Change* operator subscribes to this source, and publishes events to the application only when it decides that the mobile device has changed location. If instead, the location-sensing system followed the second approach, with the mounted sensors in the room detecting the devices in their space, the location-sensing operator graph would look like Figure 2. A room source, like Room A in this example, would publish events to which the *Location Change* operator subscribes. This operator would maintain a list of the locations of various mobile devices. When it finds that a device's location has changed, it publishes an event. There are several operators listening for such events. Each filters the event stream, forwarding only those events referring to a specific user or device.

We might need to integrate two location-sensing systems to get information about location when the user is outdoors and indoors. To do this, we would add an operator which aggregates the location information from the two systems and sends the Location Change operator the location represented in a form it can understand. The architecture can thus be modified to integrate the second location-support system without modifying the reminder application.

5.2.4 Support for Errand Reminders

SmartReminder allows the user to maintain a list of miscellaneous errands to do when she has the time. Many such "to do" items are errands to run at a particular location, such as a store. SmartReminder conveniently reminds the user about each of these items when it decides that the user has enough time to run the errand. To support this feature we have an *Errand* operator that publishes an event to the reminder application telling it that there is an errand to be run. This operator could either send the application a list of possible errands to run and let it decide which ones to display, or send only the errands for which it knows the user has sufficient time. If the operator is aware of both location and schedule information (see Figure 1) it can discern whether there is sufficient time to run an errand. It notifies the user — with an event — that an errand can be run. The operator also sends an event to the application when there is no longer enough time to run an errand. If, instead, the operator is only aware of location information, it will not be able to make a decision on whether there is time to run an errand. It sends an event to the application giving information about the errand, including the distance from the user's current location. Using this information along with the user's schedule, the application can determine if it is appropriate to alert the user. The former approach (in Figure 1) minimizes the computation on the mobile device. It means, however, that the schedule information is stored twice. This aspect of the design is discussed further later.

5.2.5 Distance Service

Solar is designed for unidirectional flow of event streams in the event graph of its publish-and-subscribe framework. Applications might, however, need to use services that take parameters and return information. These services do not simply provide information regarding their own

state as a sensor would. In the case of SmartReminder, we need a mechanism for computing the duration of the commute from one location to another. We could do this computation on the mobile device, but since we are interested in reducing the computation there, we need a service that, given information about the two locations, will return the distance between them. The SOLAR framework itself does not support such a query-oriented service. Therefore, as illustrated in the Figure 1, we include a *Distance Service*. This application, external to the SOLAR architecture, listens for requests and returns its estimate about the time it takes to travel between the two locations. The Distance Service is designed to take into consideration other context information when it makes its decision about travel time. The contextual information must, therefore, be passed to the service. The figure shows one approach: the reminder application sends the necessary context information with every request to the service. Alternatively, the Distance Service could be a SOLAR application, subscribing to the same contextual information. As a service, it could listen for connections from SmartReminder that are external to the publish-and-subscribe framework of SOLAR. There are tradeoffs between the two approaches. The latter better matches the SOLAR model, where applications subscribe to events. It also means, however, that there must be one instance of the Distance Service application for each device, since it must subscribe to the context of that device.

5.2.6 Proximate Users

SmartReminder keeps track of a list of items that the user needs to remember when he meets certain people. For example, Bob might need to be reminded to wish a certain person for their birthday if he meets them. The application should remind him at the appropriate time. There are two ways in which we can sense other users who are close to us. One way would be to have the devices directly sense proximate devices. For example, Bob's handheld might have an infrared sensor to sense other devices in its vicinity. The application will know if he is with a friend, since it can detect his friend's handheld. Given such a setup, the application is designed as illustrated in Figure 1. A *Proximate User* source detects nearby users. It then publishes an event to an operator that subscribes to these events. The job of this operator is to publish the event to the application only if the application needs to be notified. SmartReminder needs notification if a person it cares about

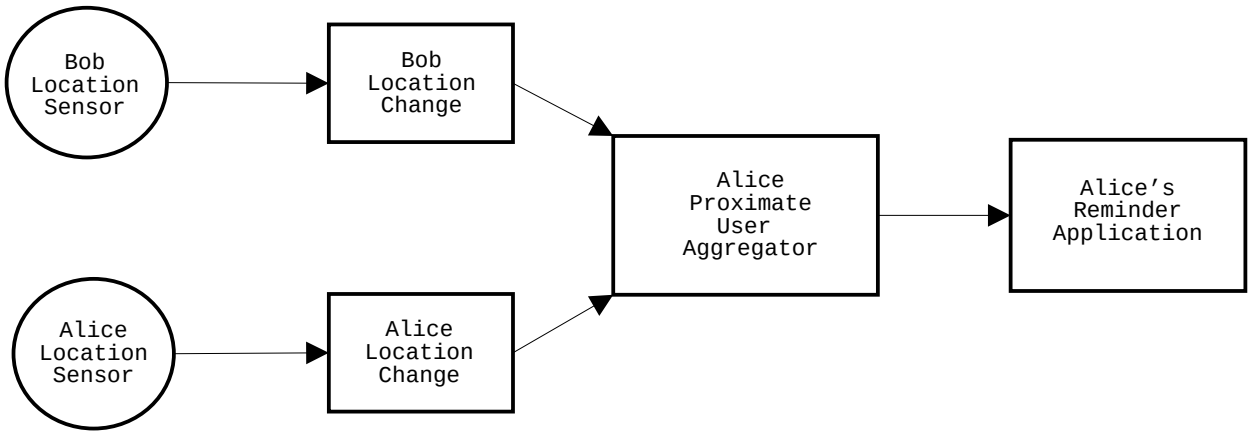

Figure 3: Objects distributed between mobile device and servers

is nearby. Therefore, the operator must only publish an event that implies a pertinent person is in the vicinity. Alternatively, proximate users can be sensed by their presence at the same location. If we use such an approach, this portion of SmartReminder would be designed like Figure 3, where Alice is interested in knowing when Bob is near. Again, this design will depend on the type of location-sensing system available.

5.2.7 Weather/Traffic Information

The application is notified of changes to weather and traffic by a filter operator. The operator is notified of the weather and traffic conditions periodically, and it decides whether the change is significant enough to notify the application. This design ignores the issue of how these contexts are sensed. Both traffic and weather are hard to quantify, and this is left for future work (Section 8). The design assumes that we have a proper way to quantify these contexts, and the operator is designed based on this quantification method.

5.2.8 Summary

The overall design needs modification based on the type of sensors available. The modifications necessary to support the different sensor systems do not involve changes to the core application, but only to the layout of the operator graph. We implemented a prototype using some of these designs to demonstrate that they are workable solutions. Section 6 discusses and demonstrates the prototype

of this application. The subsections that follow illustrate how the design of SmartReminder handles the complexities of a context-sensitive application.

5.3 Smooth Handling of Contextual Changes

The design of the SmartReminder application provides a convenient way for it to become aware of, and adapt to, changes in pertinent context. An event notifies the application of each contextual change. Operators that can discern whether information they receive must be forwarded, reduce the amount of redundant and inconsequential information the application receives. We illustrate the manner in which the application adapts easily to contextual changes with a simple scenario.

Alice uses SmartReminder. She is in her office and it will take her 10 minutes to get to her next meeting, which is in 15 minutes. She realizes, without the help of SmartReminder, that she has this meeting, and begins walking towards the venue. Enroute she meets a colleague and stops to talk. Her meeting is in 10 minutes, but from this location it would take her only 5 minutes to get to the meeting. Five minutes later, she is still talking to this person, and has not moved. So, SmartReminder pops up a reminder, and she says bye and leaves. She makes it to the meeting on time.

In terms of events, here is what happens to ensure that the application behaves as expected. When the application starts up, it reads the calendar. With the aid of the Distance Service and its contextual sensors, it estimates the best time to remind Alice of each of the upcoming appointments. It decides that it should remind her about her first appointment 5 minutes from now. It makes this decision because it knows the appointment is in 15 minutes and the user is 10 minutes away from the meeting location. As long as Alice remains in her office, the location sensor will continuously transmit the office as her location. The operator that filters out redundant information will wait until she leaves the office before sending a location-change event to the application. This event causes the application to modify its reminder plan. When Alice meets her colleague she is 5 minutes away from the meeting location and the meeting is in 10 minutes. The application knows her current location, so will set the reminder for 5 minutes from now. She talks to the colleague

and no notification of a location change is sent to the application. Since there is no contextual information being sent to the application, it does not modify its reminder time for the next 5 minutes. She continues talking, and, 5 minutes later, the application knows that it is time to remind, so up pops the reminder.

Consider instead that Alice does not meet the colleague. The application never pops up the reminder, because Alice is always going to be at the meeting on time. The location-change events cause the application to modify the reminder time as the user walks closer.

Now, suppose Alice forgot about the appointment and started walking in the opposite direction. Given the same flow of events, the application modifies its reminder time based on location, and realizes that the reminder time needs to be earlier than previously calculated.

This simple scenario illustrates the ease with which the application adapts to contextual changes. In a similar manner, the application can adapt to each of the different types of pertinent contextual information.

5.4 Handling Conflicting Contextual Information

The application uses information that could potentially conflict. It adapts to location and also to the user's schedule. The calendar might state that the user is supposed to be at a certain location. The location sensors might, however, report otherwise. SmartReminder is expected to base its reminder decision on the location of the user. The design supports this requirement. Initially, the application sets up reminders for each item in the schedule based on the location of the prior appointment. It assumes the user's location before he leaves for an appointment is the location of the previous appointment. If the calendar changes, the application gets an event notifying it of the change. SmartReminder recalculates the reminder times for appointments that are affected by the change, again based on the location of the previous appointment. The appropriate time to remind for the first meeting is decided based on current location. After the first meeting, the application has a reminder time for the next meeting computed using the location of the first

meeting. If the user remains at the location of the first meeting, this reminder time is correct. If the user moves somewhere else, however, the reminder time for the next meeting should be based on the new location and not on the location of the previous meeting. The application gets a notification of the location change, and hence computes the reminder time using the new location. It will, therefore, correctly give the location information it receives, precedence over the calendar information. Applications designed in this manner will be notified of all pertinent information in a timely fashion, allowing them to decide how to adapt, and which information should take precedence in the event of a conflict.

5.5 Using Network Resources

This section is devoted to two of the challenges listed in Section 2. To compensate for less powerful mobile devices, the application should make use of remote systems to perform some of the computation. These devices, however, use the wireless network, and might occasionally lose their connection to the network or to their sensors. Although such an occurrence limits the accuracy of the application, the application should still be designed to handle such an event gracefully, and work as best it can.

5.5.1 Handling Minimal Computing Power

Our design avoids excessive computation on less powerful mobile devices. First, the use of the Distance Service relieves the device from having to compute travel times. This calculation requires extensive map data. Second, since the application as a whole consists of sources and operators, some of these objects can run on a remote machine instead of on the handheld. For example, the *Errand* operator need not be on the mobile device. Since it runs on a remote machine, and only publishes events it knows are important, the load on the mobile application is reduced. The same applies to other objects. Figure 4 shows how the objects might be distributed between the mobile device and remote machines.

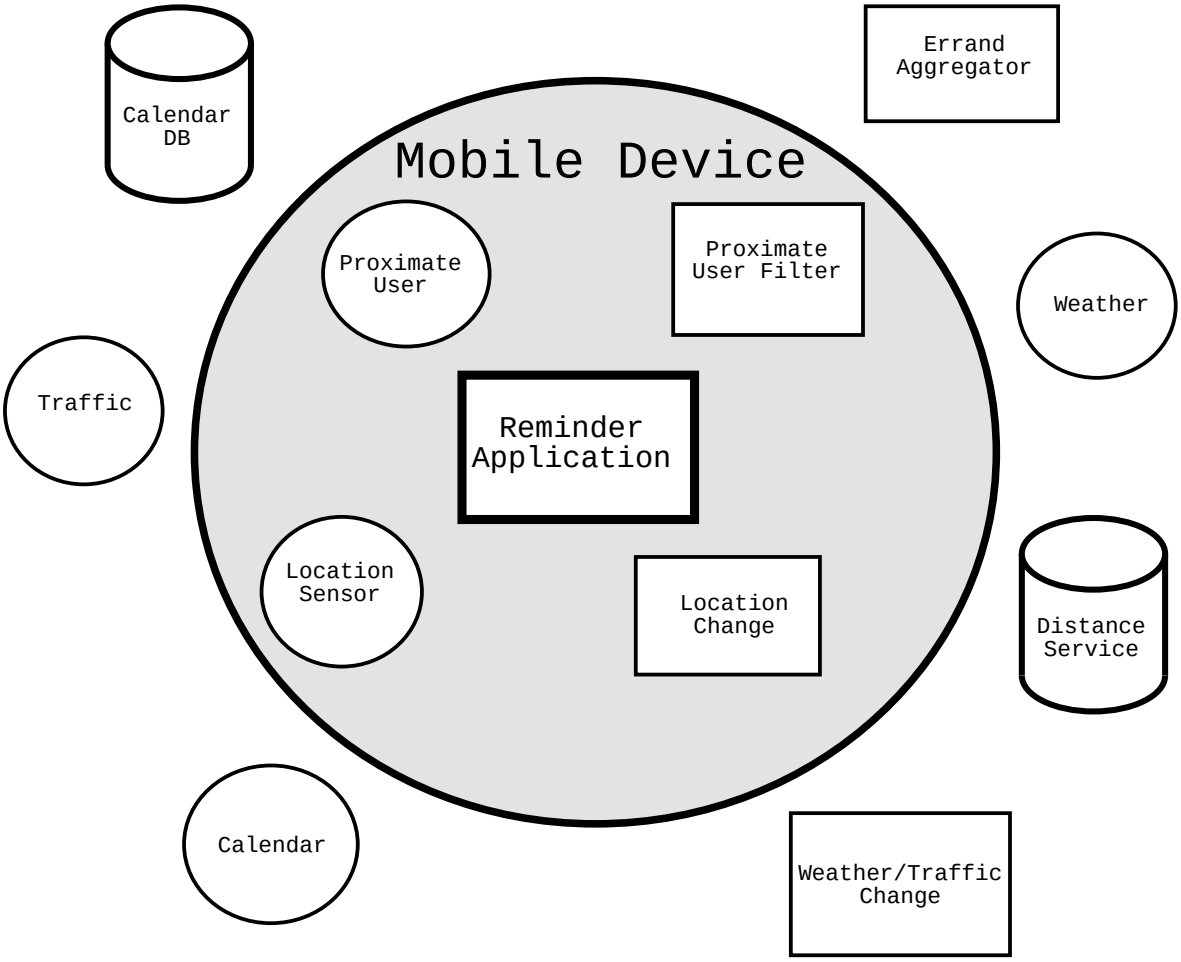

Figure 4: Objects distributed between mobile device and servers

5.5.2 Handling Network/Sensor Interruption

The application maintains state information, so if it is disconnected from its sensors, it will continue to function based on the information it received earlier. If disconnected from its location sensor, the application will remind the user based on the current calendar and last available location. Since the reminders are set up initially with calendar information, and no new location information is received, the application will automatically use the calendar, and hence handle the disconnection from the location sensor gracefully. Admittedly, the reminder might not be correct, but the application has no way of knowing the location of the user. The application may allow the user to enter their current location. When sensor information becomes available again, it will automatically be used because the application will receive an event.

Network interruption means that the application does not have access to the Distance Service. This interruption is a problem, because the application now cannot make use of contextual information even if it gets it. If this happens, SmartReminder either picks an arbitrary time, say 10 minutes before the meeting, to remind the user, or it pops up a dialog asking the user when they would like to be reminded about the meeting. It decides which to do based on user preferences. The application always first attempts to make the network connection, so it will make use of the network if it is available.

Both these interruption-handling mechanisms ensure that the application continues to function as best it can with the information it has. They also allow for the immediate resumption of the service once the connection to the network/sensors is restored. All context-sensitive applications must handle an interruption from network resources or sensors in a way that ensures that they resume regular functioning when the interruption ceases.

5.6 Compensating for Uncertainty in Location Information

Location-sensing systems attempt to accurately identify the location of the user. They are, however, not always successful. Systems like Nibble return the user's location as a list of probable

location	A	B	C	D	E
probability	.10	.15	.50	.20	.05
minutes away	3	5	8	12	16

Table 1: Probability of being a certain time from meeting at remote location, given the current location is A,B,C,D,or E.

locations with the probability of being at each of these locations. Context-sensitive applications that depend on location information should consider this uncertainty. SmartReminder must factor the uncertainty into the travel-time calculation. It should be able to take a list of probable locations along with probabilities and estimate of the travel-time to the destination. The application can convert the locations into a list of travel times between the location and the destination using the Distance Service. The procedure it should follow to estimate the reminder time is best explained using an example. Say the application has inputs as shown in Table 1, and that the meeting is in 15 minutes. Assume also that Alice depends on this application completely. If the application gives a reminder, then Alice will leave immediately for the meeting. If not, then she will not move from her current location. How happy will Alice be with the application? Alice will be dissatisfied if she is late for the meeting, and the later she is, the more dissatisfied she will be. Conversely, if the device tells Alice to leave too soon, she will be dissatisfied because she has to wait. Again, the earlier she is, the longer she has to wait, and hence the more dissatisfied she will be. The objective of the application is to make the user as happy as possible. We can model this objective using a cost function. The reminder time that makes the user most satisfied is the time that minimizes the expected cost.

5.6.1 Estimating Best Reminder Time with Cost Function

Let x be the number of minutes early the user arrives for the meeting. We need a cost function that penalizes the application for being either late or early. For simplicity we use the cost function $C_0(x) = x^2$. In the next subsection we present cost functions that are more suitable for SmartReminder. In this example, there will be no cost associated with reminding at a time such that the user arrives at the meeting at exactly the correct time. The application will suffer a penalty that is quadratic in the number of minutes early or late the reminder is.

The goal of the application is to remind at a time that minimizes the penalty. Alice's meeting is in 15 minutes, and according to Table 1 there is a 10%, 15%, 50%, 20%,and 5% chance that she is 3, 5, 8, 13, and 16 minutes away, respectively. If the user leaves now (because the application pops up a reminder), there is a 10% chance that it will only take 3 minutes to get there, and thus she will be 12 minutes early. The device will suffer a penalty of $12^2 = 144$ in this case. Similarly, there is a 15% chance that the user will be 10 minutes early resulting in a penalty of 100, and so on. What is the expected penalty? Let's call it F.

$$F = \sum_i p_i C_0(t - t_i)$$

where p_i is the probability that it will take t_i minutes to get to the meeting from location i. t is the amount of time before the event that we chose to present the reminder, so $t - t_i = x$. For $t = 15$,

$$F = 0.1 * 144 + 0.15 * 100 + 0.50 * 49 + 0.20 * 4 + 0.05 * 1 = 54.75$$

This is certainly not the best we can do. A later reminder will probably be better. The application must decide the optimal time to remind. Similarly, using t, we substitute into F and use the derivative with respect to t to find the t that will minimize F and hence be the optimal amount of time before the meeting to display the reminder:

$$\begin{aligned} F(t) &= 0.1 * (t-3)^2 + 0.15 * (t-5)^2 + 0.5 * (t-8)^2 + 0.2 * (t-13)^2 + 0.05 * (t-16)^2 \\ F'(t) &= 0.2 * (t-3) + 0.3 * (t-5) + 1.0 * (t-8) + 0.4 * (t-13) + 0.1 * (t-16) \\ &= 2.0 * t - 16.9 \\ t &= 8.45 \end{aligned}$$

Using this approach, SmartReminder can estimate the appropriate reminder time given a list of possible locations and the probability that the user is at each of these locations.

5.6.2 Cost Function for SmartReminder

The reminder application needs a cost function that penalizes both early and late reminders. For most people, however, being late for a meeting is much worse than being early, so late reminders

should suffer a higher penalty than early ones. For this reason, a cost function shaped like the one in Figure 5(a) is appropriate. Its cost grows linearly with each minute early, but grows quadratically with each minute late. The assumption is that a positive x implies an early reminder and a negative x implies a late reminder. Now that we have a graph of a function that we think is suitable, the next step is to find a function that behaves in a similar manner. We have identified two functions that behave like Figure 5(a). The first is

$$C_1(x) = S(x) * ax + (1 - S(x)) * bx^2$$

where a and b are constants and $S(x)$ is a sigmoid function that approximates a step function.

$$S(x) = \frac{1}{1 + e^{-cx}}$$

$S(x)$ is shown in Figure 5(b). It estimates a function that is 0 when x is negative and 1 when x is positive.

$C_1(x)$ has two terms. When x is positive, since $S(x)$ is 1, $C_1(x)$ is linear in x, and, when x is negative, $S(x)$ is 0, and $C_1(x)$ is quadratic in x. We can use this function in the place of $C_0(x)$ from the previous section. Using $C_1(x)$ in $F(t)$, where t is the number of minutes before the meeting to remind, we again find the t that minimizes $F(t)$ by taking the derivative.
Another possible cost function is

$$C_2(x) = a * (x + |x|) + b * (x - |x|)^2$$

Again, when x is positive, $C_2(x)$ is linear in x, and when x is negative, $C_2(x)$ is quadratic in x. Again, the constants a and b can be adjusted. We can use $C_2(x)$ in $F(t)$, but since this function has absolute values we cannot directly take the derivative to find the optimal value of t. Here is an alternate method to find the reminder time that minimizes the expected cost.

Consider the example from Table 1. We substitute into $F(t)$:

$$F(t) \quad = \quad 0.1 * (a * (t - 3 + |t - 3|) + b * (t - 3 - |t - 3|)^2)$$

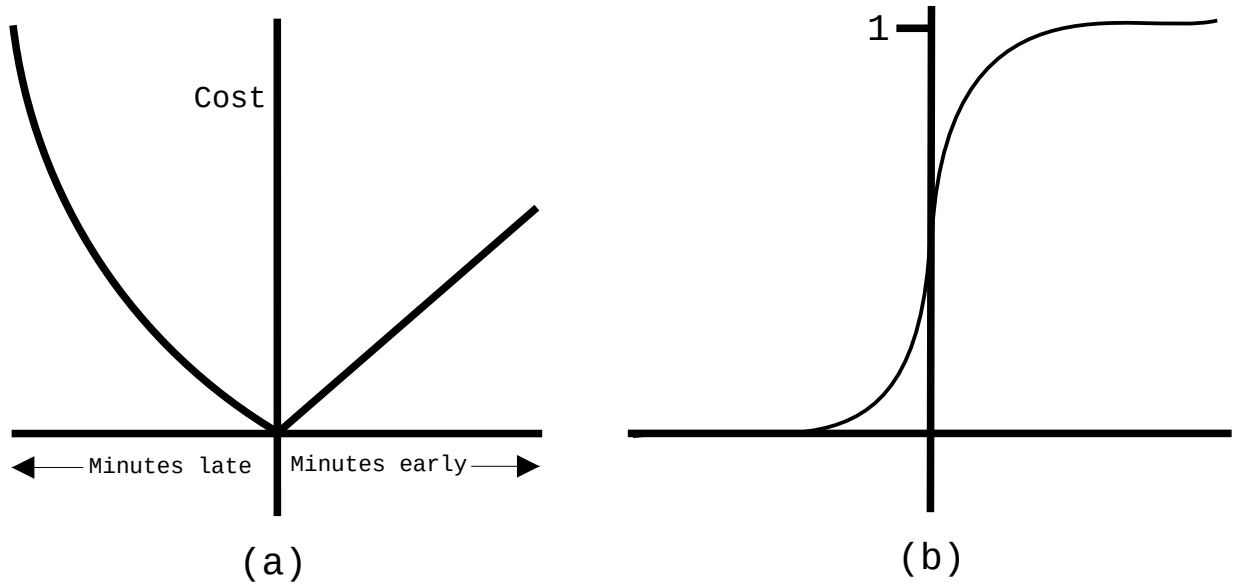

Figure 5: (a) Cost function for Reminder Application (b) Approximate Step function

$$
\begin{aligned}
&+ \quad 0.15 * (a * (t - 5 + |t - 5|) + b * (t - 5 - |t - 5|)^2) \\
&+ \quad 0.5 * (a * (t - 8 + |t - 8|) + b * (t - 8 - |t - 8|)^2) \\
&+ \quad 0.20 * (a * (t - 13 + |t - 13|) + b * (t - 13 - |t - 13|)^2) \\
&+ \quad 0.05 * (a * (t - 16 + |t - 16|) + b * (t - 16 - |t - 16|)^2)
\end{aligned}
$$

The objective is to find the value of t that minimizes $F(t)$. Figure 6 is a graph of $F(t)$ with $a = 3$ and $b = 1$. Clearly, $F(t)$ is minimized for $3 \leq t \leq 16$, where 3 and 16 are the shortest and longest possible travel times according to the Distance Service. When t is less than 3, F(t) grows quadratically, because the linear terms cancel out. Similarly when t is greater than 16, all the quadratic terms cancel out, hence F(t) grows linearly from $t = 16$ and has a positive slope.

The key to our solution is to note that if we pick values of t in certain regions, we can eliminate the absolute values. In this example there are 4 regions to consider. These are values of t between 3 and 5, 5 and 8, 8 and 13, and 13 and 16. In the third region ($8 \leq t \leq 13$), for example, we simplify

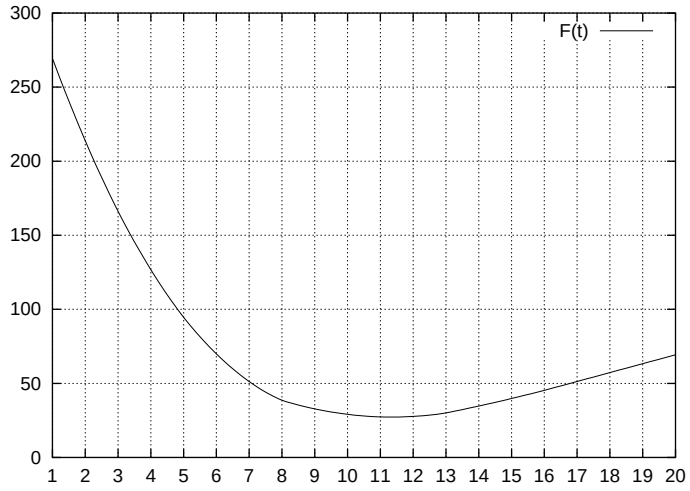

Figure 6: Graph of F(t)

$F(t)$ to $G_3(t)$, after evaluating the absolute value terms based on t being between 8 and 13.

$$\begin{aligned} G_3(t) &= (0.1 * 2a * (t-3)) + (0.15 * 2a * (t-5)) + (0.5 * 2a * (t-8)) \\ &+ (0.20 * 4b * (t-13)^2) + (0.05 * 4b * (t-16)^2) \\ G_3'(t) &= 0.2a + 0.3a + a + (1.6b * t) - (1.6b * 13) + (0.4b * t) - (0.4b * 16) \\ &= 2b * t - 27.2b + 1.5a \end{aligned}$$

Figure 7 graphs the functions for each of the regions above. In each region i, the curve for $G_i(t)$ matches the curve for $F(t)$. If a region i contains the minimum point for $G_i(t)$ then the minimum for $F(t)$ is at that point. This is true because the curve for $F(t)$ has a single turning point. Since the region contains a turning point for $G_i(t)$ and the curves match in the region, $F(t)$ will also have the same turning point. The turning point for $G_i(t)$ is a minimum point, therefore, $F(t)$ has the same minimum. Similarly, no other region j, can have a $G_j(t)$ such that the minimum for $G_j(t)$ is in the region j, because this will contradict the assertion that $F(t)$ has a single turning point. Therefore, if a region i contains the minimum point for $G_i(t)$ then the minimum for $F(t)$ must be at that point.

We can use the above fact to determine the minimum value for $F(t)$. First we calculate $F(t)$ for

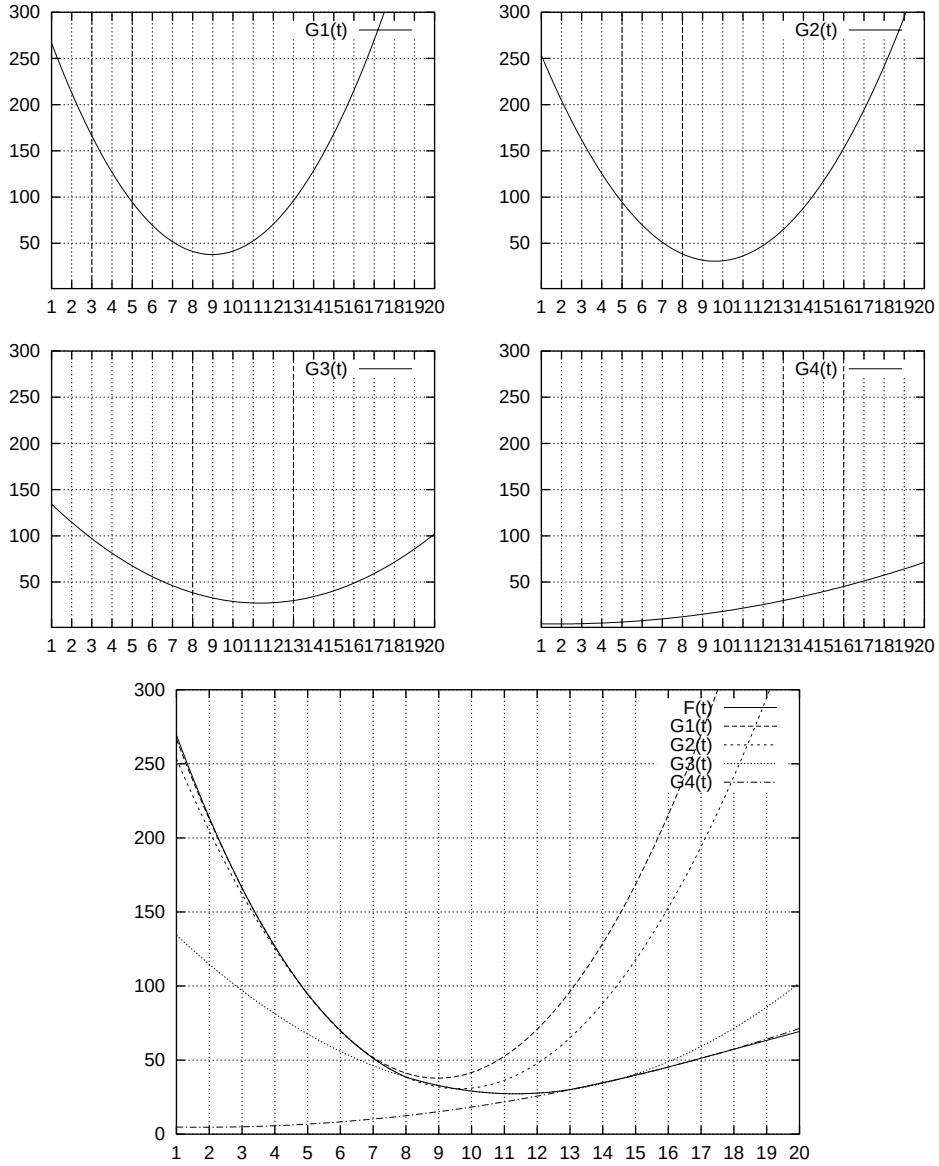

Figure 7: Shows how curves match in regions

each of the input points; the input points are the region boundaries. Next we find the value of t from this set that has the lowest $F(t)$. Since $F(t)$ has one turning point, there are now three possibilities: The minimum is in the region to the left of t, in the region to the right of t, or at t. We first remove absolute values from $F(t)$ using the region to the left. For this example, from among the 5 possible region boundaries, $F(t)$ is least when t is 13. We generate a function like $G_3(t)$ above for the region $8 < t < 13$. Then we take the derivative and solve for the value of t that minimizes $G_3(t)$. If the value is in the region $8 < t < 13$ then this is the minimum value of $F(t)$ and we are done. If not, we do the same for the region to the right, in this case $13 < t < 16$. If the minimum value is not in that region either, this is a boundary case and the minimum value is when the t is 13. For this particular example, with a = 3 and b = 1, $G_3(t)$ is minimized when $t = 11.35$. This value of t is in the region between $8 < t < 13$, so this must be the minimum value of $F(t)$. From the graph, we can see that this result is correct. In this way we can solve the cost function for the value of t that minimizes $F(t)$.

We can use either of these cost functions to estimate the optimal time to remind. Of course, other cost functions are possible.

6 Implementation of a SmartReminder Prototype

We built a SmartReminder prototype from our design. The prototype runs on Fujitsu Stylistic C-500 Pen Computers, using Windows 2000. We added a Cisco Aironet 340 Wireless LAN Adapter in the PC card slot, for access to the 802.11 wireless networks. SOLAR is implemented in Java, so SmartReminder is also implemented in Java. The location-sensing system has not yet been installed, so the prototype has a source that takes the location as input from the keyboard, and publishes the events accordingly.

6.1 Demonstration of SmartReminder

Alice arrives at work at 10:00 AM, and turns on the reminder application on her mobile computer. She is vaguely aware of her schedule for the day, but trusts her SmartReminder application to

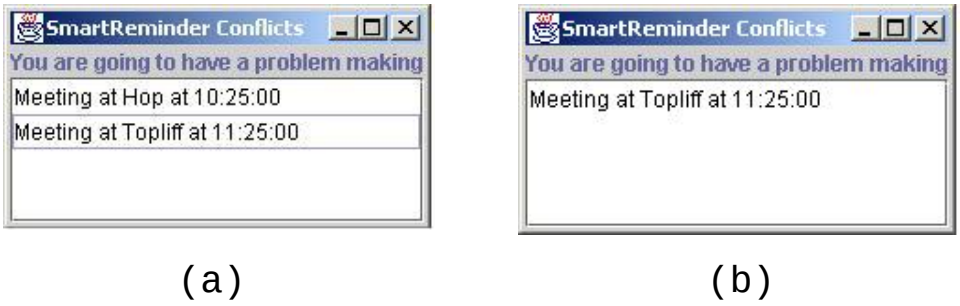

Figure 8: Screenshot of Dialog that notifies of conflicts in schedule

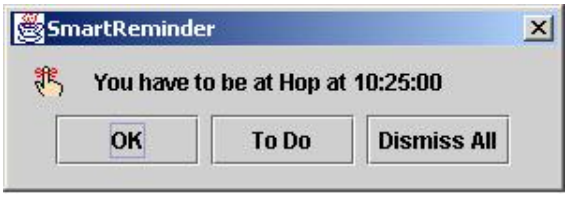

Figure 9: SmartReminder Alert

notify her of her meetings so she can make them. When the application has initialized, it displays a dialog like the one in Figure 8(a). She realizes that the second conflict is because of a meeting in her calendar that she intends to leave early. This item in the Conflicts list can be ignored. She does not know what is causing the first conflict, however. Alice looks at her calendar and sees that she is scheduled to pick up a tape from Bob at the Medical School at 10:15 AM. She does not have enough time to pick it up and still make her meeting at the Hop on time. Accordingly, Alice calls Bob and asks him to leave the tape outside his door, so she can pick it up later. She then removes the meeting from her schedule, and moves it to the list of errands she has to do when she has the time. SmartReminder updates the Conflicts screen to the one in Figure 8(b). She knows that SmartReminder will remind her to pick up the tape when she has the time.

It is now 10:10 AM and since she has just removed a meeting, she knows she has time to get something to eat before her next meeting. She walks towards the cafe, and as she enters, she sees a long line. Unsure whether she will get food or not, she decides to join the line, and wait until her SmartReminder tells her to leave. To her dismay, a few minutes later she sees the dialog shown in

Figure 9 and realizes she has to leave for her meeting if she is going to reach it on time. The meeting at the Hop is quick and finishes in 15 minutes. She decides to check if she can run an errand now. When Alice clicks the *To Do* button on the earlier reminder (Figure 9) she gets a dialog listing the errands that she can do before her next meeting, Figure 10(a). The library is nearby and she needs to return a book. She has time to run the errand, so she returns the book, and clicks the *Done* button on the dialog. The reminder to pick up the tape from Bob does not appear because there is not enough time to go to the Medical School and make it to the next meeting. She is reminded about her next two appointments, and she makes them on time. It is now noon, and once again, she clicks the *To Do* button on the last reminder. This time the reminder, as shown in Figure 10(b), tells her she can go get the tape. She goes to the Medical School to meet Bob. He is actually in his office, so Alice gets the tape and begins to talk to him. There is a beep from her mobile computer, and she looks at it to see a reminder to ask Bob how his wife is feeling. Alice then remembers that Bob's wife had surgery a week ago. Prompted by SmartReminder, she asks Bob about his wife. Alice soon gets the reminder for her next meeting, so says bye to Bob and leaves.

The example above demonstrates the functionality of SmartReminder. It is an extremely useful application for Alice.

6.2 Performance of Prototype

One of the goals when designing SmartReminder was to devise a way to easily integrate new types of contextual information. While developing the application, we added one type of context information at a time. We added operators and sources to handle the new types of context information, but made minimal modifications to the core reminder application. The events that operators publish to the application perform most of the reminder updates. Thus, we demonstrated that our design handles integration well.

The prototype adapts in a timely manner to context changes and handles failure gracefully. We ran tests with different combinations of contextual changes to ensure that the application adapted appropriately. Initially, SmartReminder seemed overly conservative in deciding on reminder time

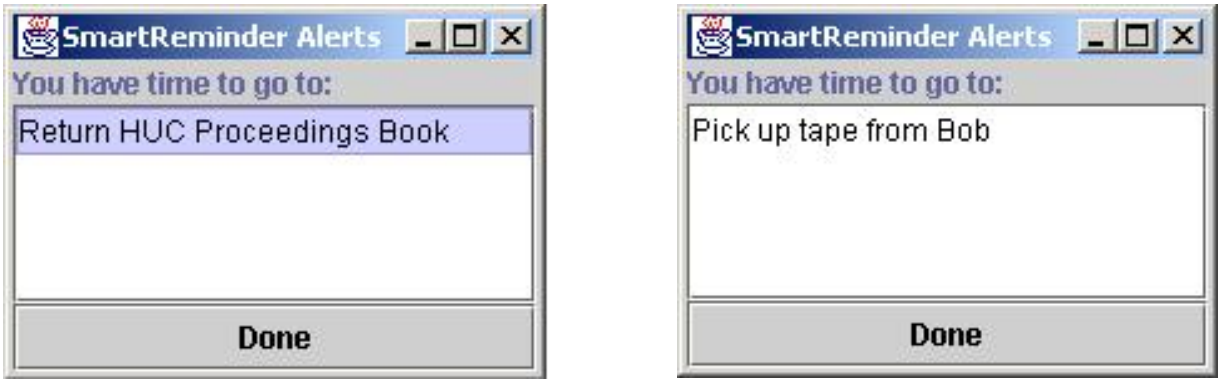

Figure 10: SmartReminder Errand Alerts

when confronted with uncertainty. It returned reminder times that were close to the farthest possible distance. We changed the constants in the cost function to make the decisions less conservative. We intend to allow the user to define these constants in preference files (see Section 8).

7 Related Work

There are two categories to consider. First, we examine systems other than SOLAR that support dissemination of context information to the applications. Second, we describe applications similar to SmartReminder that are under development.

7.1 Contextual -Information-Dissemination Systems

Many systems support transformation of raw sensor information into high-level context, which is available to applications. They are less easily expandable, because they do not provide support for applications to derive refined information from new information providers. Some systems like those below attempt to overcome this problem.

The *Context Toolkit* is a distributed system that represents each sensor with a *widget* [1]. A context widget is similar to a GUI widget. As in SOLAR, applications subscribe to contextual data published by these widgets. Applications deploy "aggregators" similar to SOLAR operators to transform the data before it is published to the application. It is unclear whether these aggregators can be stacked.

The *Context Information Service*(CIS) is an architecture that represents the world as objects [4, 10]. Objects include people and devices. Each object has state, and this state is determined by its context. The CIS updates the state of the objects based on the sensors. Applications can obtain the state of these objects.

7.2 Other Context-Aware Reminders

Proem is a system that supports profile-based cooperation [11]. Users can write simple rules that indicate their interests in other people. When someone close (in location) to the user has a profile that matches the interests of the user, the system alerts the user. It allows the user to identify people with more than just their name. SmartReminder is not quite as expandable when it comes to nearby users. On the other hand, Proem does not support reminders based on schedules.

The *comMotion* project provides support similar to SmartReminder's reminder about errands [2]. It reminds the user when she arrives at the location of the errand. It does not, however, notify the user if she is just close to the location of the errand, and there is enough time to run the errand.

CyberMinder is a reminder system that allows the users to define the contexts when a reminder should be displayed [7]. For example, it allows Alice to set a reminder for herself when Bob is in her office. In this way it is very expandable. CyberMinder uses the Context Toolkit for dissemination of contextual information.

Each of these applications displays reminders when contexts meet a certain set of requirements. In this way they are similar to the errand feature of SmartReminder. SmartReminder's most important feature, however, is that it provides timely reminders about meetings on the user's schedule based on distance from the current location. This feature is not supported by any other reminder system and is listed in the *Future Work* section of the CyberMinder system.

In terms of design, the architecture of SmartReminder is flexible. It allows for minimal modification

to the application to support the addition of new types of context information and also different types of sensing systems. Other applications like Hippie [3], a context-sensitive nomadic exhibition guide that uses a "modeling approach", are less flexible and less generic. The generic nature of the SmartReminder's design, using SOLAR, means that a similar design could be used in designing other context-sensitive applications.

8 Conclusion and Future Work

SmartReminder is working now, but it is rudimentary. The features that need to be added are listed below:

- We do not have a location-sensing system installed and a system should be added. The application itself is ready to use location events supplied by such a system.
- We do not have support for weather and traffic information incorporated into the system. This information can have a drastic effect on travel time, and hence reminder time, so we have to devise a way to sense and quantify this type of information.
- The support for uncertainty in location information assumes that one location (the destination) is certain. Destination information is not necessarily certain. Consider the case of detecting nearby users with location. The cost model must be extended to support both locations being uncertain.
- Users should be given the ability to enter their preferences into the application. For example, Bob might want his SmartReminder application to remind him of meetings 5 minutes before he should leave, but Alice might like them at the exact time. Also, a preference could be used to adjust the constants in the cost function for deciding the optimal reminder time.

To understand how well our design or our approach to uncertain sensor readings extends to other context-sensitive applications, we need to develop some other applications. Our experience with SmartReminder demonstrated that such applications can be designed using SOLAR and gave insight into the challenges involved in designing these types of applications. We devised methods for

designing applications using SOLAR and were careful to ensure that our design decisions had generic context-sensitive applications in mind, and not just SmartReminder. We, therefore, believe that we will be able to easily develop other applications if we follow an approach similar to the one we used in SmartReminder.

9 Acknowledgements

First and foremost, I thank my advisor, Professor David Kotz for his guidance throughout the course of this project. He made sure I was on track and was always available when I needed assistance. I also thank Guanling Chen for the prototype of SOLAR and for helping me design SmartReminder. Furthermore, I thank Professor Javed Aslam for discussing the uncertainty aspect of this thesis with me, and providing valuable insight. Finally, I thank my friends and family who helped me proof read this paper and listened to me discuss my thesis.

References

[1] Daniel Salber, Anind K. Dey and Gregory D. Abowd. The Context Toolkit: Aiding the Development of Context-Enabled Applications. In *Proceedings of CHI'99, Pittsburgh, PA, May 15-20, 1999*, ACM Press.

[2] Natalia Marmasse and Chris Schmandt. Location-Aware Information Delivery with *ComMotion*. In *Proceedings of Second Symposium on Handheld and Ubiquitous Computing*, pages 157-171, Bristol, UK, September 2000. Springer Verlag.

[3] Reinhard Oppermann and Marcus Specht. A Context-Sensitive Nomadic Exhibition Guide. In *Proceedings of Second Symposium on Handheld and Ubiquitous Computing*, pages 157-171, Bristol, UK, September 2000. Springer Verlag.

[4] Jason Pascoe. Adding generic contextual capabilities to wearable computers. In *Proceedings of the Second International Symposium on Wearable Computers*, Pittsburgh, Pennsylvania, October 1998, IEEE Computer Society Press.

[5] Bill N. Schilit, Norman Adams and Roy Want. Context-Aware Computing Applications. In *Proceedings of the Workshop on Mobile Computing Systems and Applications*, pages 85-90, Santa Cruz, CA. IEEE Computer Socienty Press.

[6] Mari Korkea-aho. Context-Aware Applications Survey. <http://www.hut.fi/~mkorkeaa/doc/context-aware.html>

[7] Anind K. Dey and Gregory D. Abowd. CybreMinder: A Context-Aware System for Supporting Reminders. In *Proceedings of Second Symposium on Handheld and Ubiquitous Computing*, pages 172-186, Bristol, UK, September 2000. Springer Verlag.

[8] Nissanka B. Priyantha, Anit Chakraborty and Hari Balakrishnan. The Cricket Location-Support System. In *6th ACM/IEEE International Conference on Mobile Computing and Networking*. Boston, MA. ACM.

[9] Paul Castro, The Nibble Location System. <http://mmsl.cs.ucla.edu/~castop/nibble.html>

[10] Jason Pascoe, Nick Ryan, and David Morse. Issues in Developing Context-Aware Computing. In *Proceedings of First Symposium on Handheld and Ubiquitous Computing*, pages 208-221, Karlsruhe, Germany, September 1999. Springer Verlag.

[11] Close Encounters: Supporting Mobile Collaboration through Interchange of User Profiles. In *Proceedings of First Symposium on Handheld and Ubiquitous Computing*, pages 171-185, Karlsruhe, Germany, September 1999. Springer Verlag.

[12] Guanling Chen and David Kotz. Managing Pervasive Information Sources in the SOLAR System. Submitted to *Ubicomp 2001*, April 2001.