Enabling Deliberate Design for Energy Management in Pervasive Systems

Angela Dalton and Christine Julien
Department of Electrical and Computer Engineering
The University of Texas at Austin
Austin, TX 78712
adalton,c.julien@mail.utexas.edu

Carla Ellis
Department of Computer Science
Duke University
Durham, NC 27708
carla@cs.duke.edu

***Abstract*—This paper argues for explicit consideration of data fidelity during development of context-aware systems. Increasing the amount of data captured, stored, and distributed does not always translate into into increased fidelity, and we can leverage this to avoid unnecessary energy overhead and increase device battery life. We introduce Context-Awareness Fidelity Expression (CAFE) as a framework for deliberately specifying application data fidelity adaptation with the aim of using context-awareness to reduce energy consumption.**

I. Introduction

Mobile devices and wireless sensors often provide the user interface, sensing, and data collection infrastructure for context-aware and ubiquitous computing systems. Mobile devices continue to advance in their computing power, storage capacity, display resolution, display brightness, network interfaces, and sensing capabilities. There is also a trend toward smaller and more compact mobile devices. While these advances make devices more appealing for consumers and more able to provide rich functionality, improvements in battery lifetime have been limited. Even though new devices are designed using low-power electronics, they have only a finite, highly constrained battery life.

We address this challenge through careful consideration of data fidelity, meaning the accuracy of the representation of true context, in terms of the amount of data captured, stored, and distributed by context-aware systems. Increasing the amount of data necessarily increases energy consumption, however it is not as simple as assuming that the most or least amount of data, which in turn results in the highest or lowest average energy consumption, provides the greatest or lowest level of fidelity. To this end, we have developed the Context-Awareness Fidelity Expression (CAFE) framework to guide reasoning about the energy/data fidelity relationship in context-aware applications, which can greatly benefit through better accuracy and lower energy overhead.

We begin this position paper with a discussion of context data fidelity in three dimensions: capture, storage and distribution. We overview related work in the next section. In Section IV we describe CAFE, our approach to directly managing data fidelity in context-aware systems. Section V summarizes and concludes.

II. Related Work

Our work fits in with a variety of tools that exist to aid in the development of context-aware applications. These tools include middlewares, prototyping environments, modelling languages, and programming libraries.

Perhaps the most well known development aid, The Context Toolkit [6] provides abstractions for building context-aware applications that allow the use of context information without requiring direct control over context sensing. Context Modelling Language (CML) [11] is a tool that assists a developer in formally specifying the context requirements of an application, including required context information and sources for that information. The Java Context-Awareness Framework (JCAF) [1] is a programming API for developing context-aware applications. The Sentient Object Model [7] provides abstractions for using sensor information, representing it as context, and actuating objects based on the context. A framework for developing context-aware applications based on the sentient object model is presented in [2].

Many efforts exist to develop context-aware middlewares. The CAMPUS middleware [9] targets applications that will run on PDAs, phones, and wearable devices. CAMPUS provides a rules system for deriving context following events. CARISMA [3] uses reflection to adapt to context changes and resolve conflicts that arise when peers have different context requirements. Another approach, introduced in [13], is the use of design patterns for improving the quality of context-aware applications.

None of these tools specifically address the requirements for context data fidelity, which differentiates our work. We believe that our work could be used in conjuction with other development tools to provide structure specific to data fidelity considerations in context-aware applications.

III. Context Data Fidelity

Characteristics of context data can be quantified in terms of context data fidelity. We define context data fidelity as the level of representation provided by context data with respect to a specified reference, which can be considered as the "truth" that the context data are supposed to determine. Context data fidelity is in effect an error metric. Assigning a value to context data fidelity is done by first defining the reference that the

context data are to determine. The context data fidelity level is then the statistical error between the context result determined from the context data and the defined reference.

It has been a widely accepted hypothesis that there exists an increasing relationship between the amount of data used to determine context state and the context fidelity level, i.e., more data provides higher accuracy. Due to the heterogeneous nature of context sources, however, this is not always the case for context-aware systems [5]. For example, if we are using a camera to collect images to detect presence of a person, the image resolution must be high enough to perform the detection, but should it be set too high, people in the distance might cause false positives for presence detection. We address context data fidelity in three dimensions: capture, storage, and distribution.

A. The Capture Dimension

Context-aware systems designed to run on mobile devices rely to a large degree on data collected by sensors to determine context. Many different sensors can be used, and these vary widely in the type, amount, and precision of information they capture. Parameters characterizing sensor data capture, including individual as well sets of sensors, form the *capture dimension* of context data fidelity.

B. The Storage Dimension

Data about past context can provide information for prediction of future context or analysis of current context. Such prediction or analysis can be done both by detecting patterns of context or by extrapolating new context either entirely or partially from recent contextual data. Obviously it is necessary for some context data to be retained in some form and for some lifetime by the system to enable the capabilities of many context-aware applications that use historical data. Finally, some context-aware applications include historical data or a logging mechanism as a fundamental component of the service they are designed to provide. Parameters characterizing the retention of data form the *storage dimension* of context data fidelity.

C. The Distribution Dimension

Due to the resource limitations of mobile devices, context-aware systems that run on them often rely on wireless communication with external infrastructure to offload intensive computing tasks as well as for expanded data storage. Sensor data can also be collected by sensors external to the mobile device and transmitted to the device. Externally collected sensor data could first be analyzed by external infrastructure components of the system, and then some set of results might be sent to the device for use by the system. Context-aware systems that involve sharing context data with other users or with a context-aware infrastructure necessarily require distribution of some context data. Parameters characterizing how data is distributed comprise the *distribution dimension* of context data fidelity.

D. Fidelity and Energy Consumption

The adaptation of what has been referred to as data fidelity has been one of the most important techniques employed in energy-aware software design [8], [12], [4]. Reducing fidelity has been traditionally viewed as a way to reduce the amount of work required and, consequently, the energy consumed to deliver a service. However, for context-aware applications, increased amounts of data capture, storage, and distribution do not necessarily result in increased context data fidelity. Instead, there can be a sweet spot, after which context data fidelity either does not improve or even degrades. Energy consumption is non-decreasing as the amount of data captured, stored, and distributed increases due to a necessary increase in usage of energy consuming resources to perform the capture, storage and distribution of the context data. This can be used to improve the energy consumption characteristics of context-aware systems.

IV. CAFE

As we discussed in the previous section, careful consideration of the amount of data captured, stored, and distributed by context-aware systems can be useful in optimizing context data fidelity and system level energy consumption. In this section we present a framework for reasoning about data fidelity as it relates to the amount of data captured, stored, and distributed in context-aware systems.

Our framework, called Context-Awareness Fidelity Expression (CAFE), can be used as a tool by developers to assist with specification of application behavior. CAFE is a method for describing the capture, storage, and distribution dimensions that, when combined, determine data fidelity in a context-aware system. It can also be incorporated into an application through the use of a set of Java objects we provide that allow application developers to ensure data resolution or granularity consistent with the specified requirements of the application. By using *CAFE enablement objects* in coding their applications, developers have a defined structure for incorporating fidelity related behavior, and making later changes easier to implement. By incorporating the CAFE ideas, system energy consumption overhead caused by working with excessive amounts of context data can be reduced without compromising the effectiveness of the service provided.

A. The CAFE Method

Context-Awareness Fidelity Expression provides a procedural method for specifying the behavioral requirements of context sources in context-aware systems with respect to three dimensions: capture, storage, and distribution of sensor and context data. Expressing sensor and context data fidelity requirements of context-aware systems in terms of the amount of data captured, stored, and distributed by context-aware systems gives developers a way to understand and compare characteristics of the system relating to energy consumption.

Context data fidelity refers to the level of accuracy of representation with respect to a reference. Of course, the relevant features of the real world to which the sensor and context

data are compared, the reference, are application-specific and must be specified for this to become meaningful. The CAFE procedure consists of four steps that provide the basis for specifying the CAFE dimensions in a particular system. Each step builds on the previous one, and the procedure taken as a whole is used as a guide for designing the behavior of an application with respect to capture, storage, and distribution of context data. The initial step in the CAFE procedure is to list the high-level *context types* that are used by the system to influence its behavior. The *context states* determined by each context type correspond to the defined reference by which fidelity is measured.

The next step is to inventory the *context sources* available to the system. Context sources include individual sensors as well as system properties or system data sources relevant to the context-aware application for which the expression is being developed. The context sources must be assigned parameters that indicate the amount of data the source is providing to the system. Parameters are specific to each context source and represent attributes such as resolution, noise level (which would have a negative value), acquisition frequency, field of view, and many others. The parameters may have a specific value, a range of possible values, or a set of possible values.

Next in the procedure is a pairing step, which involves matching context types with the context sources used to determine them. Multiple pairings can exist for a given context type, and the pairing can match a context type to a tuple of context sources that contribute to analysis of that context type. Sources in the tuple can be either required or optional. The pairings correlate the high-level context types with the individual context sources or aggregation of sources from which data can be collected and analyzed to determine their state.

A pairing in which specific context sources can be optional, for example, might be a Location context type paired with the context sources of GPS and Wireless Radio. The Location state provided by this pairing might be determined by taking the midpoint between the coordinates provided by the GPS and Wireless Radio context sources. However, if one or the other method is unable to provide a fix, the Location would be based solely on the available source.

As the final step in the CAFE procedure, a *context value specification* is created for each pairing. The data fidelity values assigned to context type data should approximately reflect the level of representational accuracy to which the context can be determined by the set of sources in the pairing. Specifications used to approximate the level of fidelity have different properties based on the context type and characteristics of the context sources in the pairing.

For example, in a pairing with location as its context type and where the context source is a GPS receiver, the context source data from the GPS can be expected to provide location within 3 meters of actual location if at least 4 satellites are visible. When 3 satellites are visible, only a 2D position can be determined. When fewer than 3 satellites are visible, a location cannot be determined, and the data value would be considered negligible [10]. Pairings and specifications are application-specific, therefore a pairing of the same context type and sources might have different specifications for different applications. The process of creating a context value specification for an application is the most involved step of CAFE, which we will summarize herein. A detailed explanation and examples for each dimension can found in [5]. This step is comprised of specifying attributes in each of the three CAFE dimensions.

The context value specification for a pairing's capture, storage, and distribution dimension defines a value for each CAFE Parameter specific to the context sources in the pairing.

To provide a relative valuing among pairings or a relative valuing among similar applications, we quantify each dimension through a summation of the parameters of the context sources. In the case of the storage dimension, we apply a reduction factor, rho_s, to the source parameters, representing the degree to which the data stored is reduced from that captured. We also apply a lifetime variable, tau, incorporating restrictions on storage duration. In the case of the distribution dimension, we apply a reduction factor, rho_d, for the same reason, and a freshness variable, phi, representing a loss of fidelity as data ages. We use a summation of normalized parameters as a simple way of accounting for heterogeneous context sources and their parameters with equal weight.[1] The minimum possible value for the capture dimension is then the total of least possible values for all parameters and the maximum possible value is the total of highest possible values for all parameters. The lower bound for capture could be zero if every context source had zero as a possible value. Although a context-aware application would seem by definition to be unable to function without any context information, instead of the capture dimension providing context information locally, it could be obtained through the distribution dimension, for example receiving context data from a source external to the system. Even in the case of an application in which capture is required for obtaining context information, zero as a lower bound indicates that no single context source alone is critical to the system functioning; instead alternative sources may be used. Examples of this include the ability to use either RFID tag data or face recognition from images captured to determine identity, and the ability to use either GPS or WiFi-based location sources. In these cases, only one source is required, though both might be available.

Once all of the steps of the CAFE procedure are complete, one can quantify the fidelity behavior of an application as the vector (c, s, d), where c is the capture value, s is the storage value, and d is the distribution value.

B. CAFE Enablement

CAFE can be used by developers as a tool for reasoning about and appropriately specifying data fidelity requirements and corresponding capture, storage, and distribution settings.

[1]More complex methods than summation might better represent capture values, but we want to provide simple initial comparitive values.

It can also be used as a method for estimating and comparing relative energy consumption characteristics of context-aware systems. Finally, CAFE can serve as the basis for enforcing data fidelity behavior in each of the fidelity dimensions when incorporated into context-aware systems using the Java CAFE Enablement Environment. The CAFE Enablement Environment is a Java-based implementation of the CAFE framework that can be incorporated into a context-aware system to provide enforcement of the characteristics and settings defined by the application, as well as providing a structure for a developer to incorporate fidelity settings into the application. The implementation includes classes for the components described in the previous section as well as a *CAFE Manager (CM)* class. The CAFE Enablement Environment uses the Java event model as the basis for handling capture, storage, and distribution of sensor and context data. The CM also initiates updates of the CAFE determined settings upon the occurance of a context change requiring re-evaluation of the settings. Again, this is handled through the use of Java events triggered by context changes.

Every sensor available in the system is represented by a Context Source object, as are other system-internal sources of context data. The capture, storage, and distribution behaviors required by the application are enforced on these objects through the CM. Applications provide the CM with pairings between context sources and types and a context value specification for each pairing.

The CM orchestrates the overall functionality of CAFE Enablement through state change listeners that monitor the states of all Context Type objects. It performs the translation using application specific Context Value Specifications and the Context Source capabilities to actual sensor settings (or access control settings for system resource context sources) to reflect the correct capture, storage, and distribution dimension values. The CM dynamically determines and controls the Context Sources and Context Types' settings using the application defined Context Value Specifications, the current context state, and the settings available for Context Sources.

The CM directly actuates each sensor's hardware settings to ensure that the application will capture only the CAFE specified amount of context data based on its Context Value Specification for the current context state. The CM controls the storage and distribution of context data by performing the translation from specified requirements to settings. In addition to controlling the storage and distribution of data from low-level context sources, such as sensor data, the CM controls the settings for high-level context information that have been determined through analysis of context source data. It provides applications with interfaces through which they can perform store and send operations on sensor data and context information. This level of indirection allows the CAFE Enablement Environment to guarantee that the data stored or disseminated matches the CAFE specifications for storage and distribution. To this end, a store request includes the lifetime allocated for the data being stored. The CM schedules the removal of the data from the file system within a short time of the item's *time of death*.

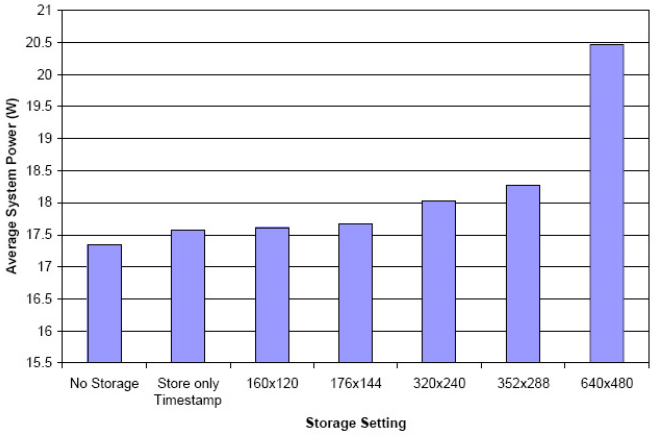

Fig. 1. Average FaceLog Power Consumption

C. A CAFE Example: FaceLog

To provide a simple and concrete example of the use of CAFE Enablement in a context-aware application, we have developed FaceLog. FaceLog is a context-based auditing tool for keeping a record of users of a shared system, logging information about the users of that system. The record contains an initial image of each user when his or her attention is first detected, stored along with the *time of initiation*. Subsequently, the corresponding *time of departure* of that user is determined and stored when his or her attention is no longer detected. These individual *sessions* provide a record of system usage for auditing purposes.

A simple and naive implementation of FaceLog might simply set the camera resolution to be able to meet all application requirements, using 640x480, the highest resolution supported by the camera, allowing user identification through face recognition. Figure 1 shows the average power when running FaceLog and storing images at each supported camera resolution. Using CAFE, we can specify the application requirements at a finer granularity, specifically that of each context state, and provide appropriate fidelity adaptation through the CAFE Enablement objects, allowing the system to stay at the lower levels of power consumption for the majority of execution.

We have implemented FaceLog both with and without the CAFE Enablement framework, using identical fidelity characteristics in both versions to provide a fair comparison. In an implementation done without specifically reasoning about fidelity requirements in each context state, it is likely that system energy consumption would be higher, especially in more complex applications than FaceLog.

We compared the standalone FaceLog implementation with the CAFE Enabled version to understand what, if any, overhead was added by incorporating the CAFE Framework. We instrumented each version of the FaceLog code to measure time for operations performed when the attention context type was in each possible state, and therefore caused different

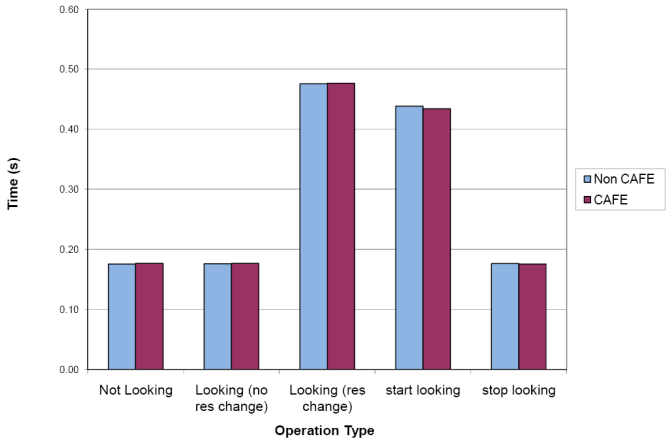

Fig. 2. Operation Times for Stand-alone vs. CAFE Enabled FaceLog

application behavior. An operation for the context state in which the user "starts looking" requires the camera resolution to be changed to the highest supported resolution (so as to enable face recognition), the subsequently captured image to be stored, and the time of initiation to be logged, which matches the image filename. Two operation types exist for the context state in which the user is "looking." In the first operation, the camera resolution is changed to the lowest supported resolution in addition to the capture and analysis of an image, since face recognition support is no longer necessary. The common case operation in this state is simply capture and analysis of the image. The "not looking" state similarly involves only capture and analysis of the image. The "stop looking" state requires storage of the time of departure. Our findings (Figure 2) show negligible performance difference between the standalone version of FaceLog and the CAFE Enabled version for each type of operation.

The behavior of both versions of FaceLog is the same, with each version optimized based on our consideration of the capture and storage requirements and the fidelity characteristics we found in our previous analysis. Clearly it is possible, then, to develop a stand alone application that uses the optimal capture, storage, and distribution settings for the data fidelity characteristics of the system. We argue, however, that the CAFE Enablement framework provides the application developer with the basic code infrastructure that guides him or her through the process of considering data fidelity requirements and corresponding specifications for each dimension. In the case of FaceLog and without the framework, it would certainly be plausible, if not likely, that a developer would simply select the camera resolution required for storage at the beginning of each session and leave the camera set at that resolution throughout execution. We developed both versions to provide the same functionality so that we could fairly evaluate the overhead of using the CAFE framework for implementation.

V. Conclusion

Energy management continues to be a primary challenge for context-aware applications on mobile devices. In this position paper we have argued that developers of context-aware systems should explicitly specify data fidelity behavior of their applications. Appropriate data fidelity specification leads to energy usage matched to the demands and state of the application, and avoids excessive overhead. In order to address the need for tools to guide users and developers in understanding and specifying fidelity requirements, we created the Context-Awareness Fidelity Expression (CAFE) Framework. CAFE provides a procedural method that can be used for specification and evaluation of data capture, storage, and distribution characteristics in context-aware systems. We created a set of Java objects that we refer to as a CAFE Enablement Environment. The CAFE Enablement Environment allows developers to incorporate CAFE specifications into their applications and ensures behavior matching the specifications. We evaluated CAFE by developing in parallel two versions of FaceLog, a context-aware application for user auditing, one using the CAFE Enablement Environment, and one standalone version. We showed that deliberate consideration of fidelity requirements can produce significant energy savings. Our comparison of the two FaceLog versions showed that CAFE did not add performance overhead.

VI. Acknowledgements

The authors would like to thank the Center for Excellence in Distributed Global Environments for providing research facilities and the collaborative environment.

References

[1] J. E. Bardram. The java context awareness framework (jcaf) - a service infrastructure and programming framework for context-aware applications. In *Pervasive*, pages 98–115, 2005.

[2] G. Biegel and V. Cahill. A framework for developing mobile, context-aware applications. *percom*, 00:361, 2004.

[3] L. Capra, W. Emmerich, and C. Mascolo. Carisma: Context-aware reflective middleware system for mobile applications. *IEEE Transactions on Software Engineering*, 29(10):929–945, 2003.

[4] S. Chandra, C. S. Ellis, and A. Vahdat. Multimedia web services for mobile clients using quality aware Transcoding. In *WoWMoM*, Seattle, WA, August 1999.

[5] A. Dalton. *Data Fidelity Mechanisms for Enhancing Energy Management in Context-aware Systems*. PhD thesis, 2007.

[6] A. Dey and G. Abowd. The context toolkit: Aiding the development of contextaware applications, 1999.

[7] A. Fitzpatrick, G. Biegel, S. Clarke, and V. Cahill. Towards a sentient object model, 2002.

[8] J. Flinn and M. Satyanarayanan. Energy-aware adaptation for mobile applications. In *SOSP*, pages 48–63, December 1999.

[9] K. Hisazumi, T. Nakanishi, T. Kitasuka, and A. Fukuda. Campus: A lightweight and dependability oriented context-aware middleware. In *The 3rd CREST/ISWC Workshop on Advanced Computing and Communicating Techniques for Wearable Information Playing*, 2004.

[10] G. Ltd. What is gps? http://www.garmin.com/aboutGPS.

[11] T. McFadden, K. Henricksen, and J. Indulska. Automating context-aware application development, 2004.

[12] D. Narayanan and M. Satyanarayanan. Predictive resource management for wearable computing. In *MobiSys 2003*, pages 113–128, New York, NY, USA, 2003.

[13] G. Rossi, S. Gordillo, and F. Lyardet. Design patterns for context-aware adaptation. In *SAINT-W '05: Proceedings of the 2005 Symposium on Applications and the Internet Workshops (SAINT 2005 Workshops)*, pages 170–173, Washington, DC, USA, 2005. IEEE Computer Society.