CONTEXTO: Leveraging Energy Awareness in the Development of Context-Aware Applications

Maximilian Schirmer[1], Sven Bertel[2], Jonas Pencke[3]

Abstract

We introduce a new context classification and recognition framework for the development and deployment of mobile, context-aware applications. The framework is complemented with an energy calculator that specifically assists mobile developers in estimating the energy footprint of context-aware applications during the development process with the framework. The framework abstracts from the raw context information gathering, allows for sensor fusion, enables the prediction of custom and higher-level contexts, and provides for context sharing.

1. Introduction and Motivation

The evolution of mobile devices and the general availability of information sources that describe the situation and environment (i.e., the context) of mobile users offer new opportunities for innovative applications [1]. By constantly monitoring the contexts in which mobile users are situated, applications obtain a potential to adapt their behaviour to current contexts more intelligently and without user intervention. However, such mobile context awareness comes at a price: Novel challenges of the mobile environment and specific constraints of mobile devices and their use (e.g., limited battery life, a comparably small screen size, dependence on network infrastructure) can severely impact the acceptance of mobile context-based approaches. In addition, adequate developer support for the realisation of context-aware applications is currently lacking. Consequently, most application developers are on their own when realising the sensing and interpreting of context information, or the sharing of context. With the increasing interest in, and a growing market for, context-aware applications, developers are more and more in charge of carefully designing context-aware applications and they need to be able to competently address issues such as privacy [2], availability, precision of context recognition, or energy requirements.

In this contribution, we address the energy-related implications of developers' choices of sensing components, processing algorithms, and granularity or temporal frequency of sensing. We specifically aim at developer energy awareness and present CONTEXTO, an energy-aware framework for offline context classification and recognition on mobile devices. The framework provides a layered, component-based architecture that can easily be extended, modified, or customised. It follows established software engineering patterns to provide high learnability and a low threshold for beginners. Within the framework, the energy requirements for all used components on a specific device are always made transparent, and information about energy requirements can be used early in the design process with the help of the framework's energy calculator, and at runtime.

The following section will introduce the main concept and energy model of CONTEXTO. Section 3 will address the software architecture and implementation details. Section 4 will give an overview of related work. Section 5 will provide an outlook on future work.

[1] Bauhaus-Universität Weimar, Germany, maximilian.schirmer@uni-weimar.de, Usability Research Group

[2] Bauhaus-Universität Weimar, Germany, sven.bertel@uni-weimar.de, Usability Research Group

[3] Bauhaus-Universität Weimar, Germany, jonas.pencke@uni-weimar.de, Usability Research Group

2. Energy Model

CONTEXTO aims at providing energy awareness for developers of context-aware applications. We hope that insights into the specific energy footprints of alternative implementations of context recognition will lead to more energy-efficient applications that, in turn, will be more widely accepted by its users. At its core, the framework employs an energy model for a number of smartphone devices. The model provides information on the individual power consumption and required energy for all of a device's sensors in relation to a chosen sampling interval [3]. We conducted extensive measurement experiments with a software-based remaining capacity approach to build energy models for the Apple iPhone 4, 4S, and 5. A detailed description of the measurement setup would go beyond the scope of this paper, in essence we read the remaining battery capacity in mAh, as provided by the *IOPowerSources* part of the *IOKit* framework (please see [4] for further details). Information about the energy demand of the current selection of sensors is made available to developers at runtime, directly within the framework.

In contrast to most of the classic context platforms and toolkits that rely on a distributed or client/server-based architecture, CONTEXTO is completely self-contained and provides true offline context classification and recognition. All steps of the processing pipeline (data acquisition, pre-processing, context classification, context prediction) happen directly on the mobile device, and there is no external service or context platform required. Our tests of the framework have shown that recent smartphone models provide the required resources (e.g., CPU speed and RAM size) for all processing steps. We think that the offline approach is superior to online approaches because it allows for higher levels of privacy (all gathered data remain on the device), reduces the amount of energy required for the context recognition (UMTS and WiFi hardware requires most of the energy on a smartphone [5]), and does not rely on any kind of network infrastructure.

Component	Energy demand (J, 10 minutes)		
	iPhone 4	iPhone 4s	iPhone 5
Sensors			
Accelerometer	29.15	25.13	20.42
Camera	217.02	317.00	293.89
GPS	143.29	143.56	180.44
Gyroscope	61.67	49.53	34.26
Heading (Compass)	69.23	70.34	37.41
Features (measurement / computed)			
Camera+GPS	543.31 / 549.63	662.45 / 657.00	638.49 / 646.16
Accelerometer+Orientation	237.68 / 240.86	247.28 / 241.93	202.97 / 205.79
Framework (local / remote)			
Baseline	211.86 / 268.18	245.12 / 318.92	181.89 / 249.30
Battery+Carrier+Contacts+Date+Reminder	302.45 / 380.18	358.31 / 486.87	307.44 / 399.47
Camera+GPS+Heading+NetworkInfo	671.80 / 750.35	846.49 / 977.99	694.60 / 808.45

Table 1: Device-specific energy demand overview of hardware sensors

We conducted an extensive series of experiments on all three devices. On each device, the energy demand of about 20 hardware and software sensors (the exact number varies, depending on the device capabilities and available sensors) and 10 features was measured with a sampling frequency of 1 Hz over the course of 10 minutes. For every sensor/feature, we repeated each trial three times on every device. We also evaluated if our initial argument for local (offline) processing and classification directly on the device had a positive energy impact, compared to a remote solution where gathered sensor data was sent to a server using WiFi. Table 1 gives a short overview of some of the most interesting findings. In the table, we compare the energy demand (expressed in Joule for a 10-minute trial, all values are averages across three trials) of selected sensors and features across our tested devices. Furthermore, we used the measured energy demand of individual sensors to approximate the energy demand of our features. Finally, we evaluated the difference between keeping all data local vs. sending data to a remote server for processing.

Our gathered data indicates that most of the sensors in the iPhone 5 (the most recent device among our test devices) show a lower energy demand than the older sensor components in the iPhone 4 and iPhone 4S. There is an exception with the GPS sensor of the iPhone 5. Since that sensor's performance has drastically improved since the iPhone 4S, we suppose that, for the iPhone 5, Apple opted for high accuracy and low latency over low energy demand.

We were also able to show that the sum of the individual energy demands of sensors very well approximates the energy demand of features using these sensors. In most cases, the computed energy demand overshoots the measured energy demand; this is probably due to internal optimisations of the underlying iOS sensor APIs.

Finally, our evaluation clearly shows that sending sensor data over a WiFi connection requires a considerable amount of energy; and that this overhead cannot be compensated with energy saved by outsourcing the processing and classification algorithms. We are still looking into this issue, since we assume that very complex processing algorithms and classifiers running on the smartphone may change this circumstance.

3. Implementation

The CONTEXTO framework follows a layered architecture, based on the proposed architectures in [6, 7]. As shown in Figure 1, it separates context usage from context detection. In the *Context Detection* layers, we set a *Managing* layer that contains context models and persistence functionality on top of classifiers and features of the *Thinking* layer. The basic *Sensing* layer contains all sensors that are used as data sources.

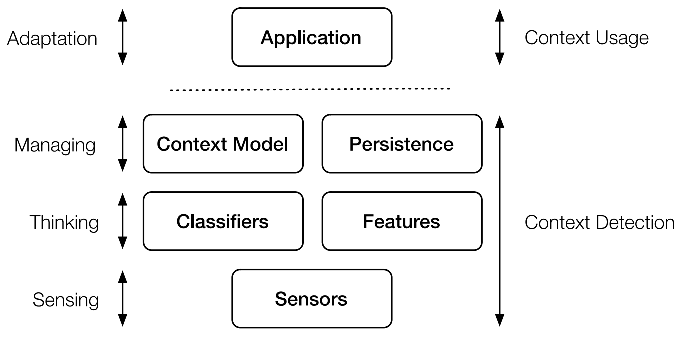


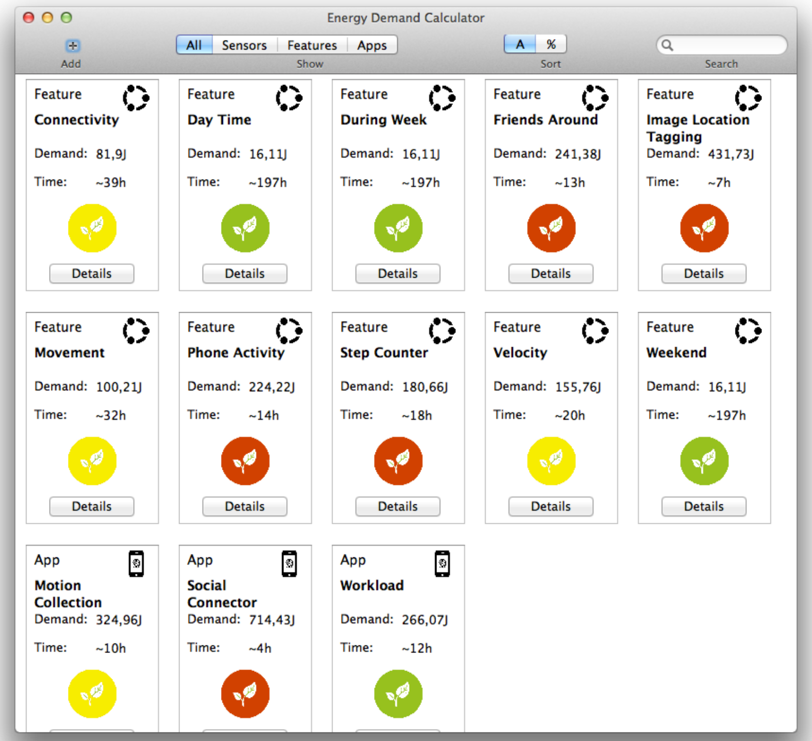
Figure 1: Layer architecture of CONTEXTO

Our prototype implementation is currently available as an iOS framework, but the concept can be adapted to other smartphone operating systems with little effort. The sandbox design of iOS currently prohibits a service architecture for providing context data to other applications, but this could easily be overcome on other platforms. The framework implementation was tailored to support easy extensibility, learnability, customisability, and maintainability by paying particular attention to accepted design principles (Separation of Concerns, Single Responsibility, Interface

Segregation, Dependency Inversion). This means in particular that developers familiar with common iOS frameworks should have a fast learning experience and quickly make efficient use of CONTEXTO. All components can easily be extended or replaced. In our prototype implementation, we currently provide a naïve Bayes classifier; additional classifiers can be added by overriding our well-documented *JPClassifier* class. All key components follow this pattern of extensibility.

4. CONTEXTO Energy Demand Calculator

The *Energy Demand Calculator* (see Figure 2) is a tool for developers to estimate additional energy demands of selected information sources (sensors and features) for context acquisition. The tool gives developers the ability to determine an approximated energy demand early in the development process of a context-aware application. The calculator allows developers to try out different combinations of information sources, and visualises their impact on the energy demand of the application. Such information allows optimising an application's energy footprint and increases the awareness for major energy consumers.



Feature
Connectivity
Demand: 81,9J
Time: ~39h

Feature
Day Time
Demand: 16,11J
Time: ~197h

Feature
During Week
Demand: 16,11J
Time: ~197h

Feature
Friends Around
Demand: 241,38J
Time: ~13h

Feature
Image Location Tagging
Demand: 431,73J
Time: ~7h

Feature
Movement
Demand: 100,21J
Time: ~32h

Feature
Phone Activity
Demand: 224,22J
Time: ~14h

Feature
Step Counter
Demand: 180,66J
Time: ~18h

Feature
Velocity
Demand: 155,76J
Time: ~20h

Feature
Weekend
Demand: 16,11J
Time: ~197h

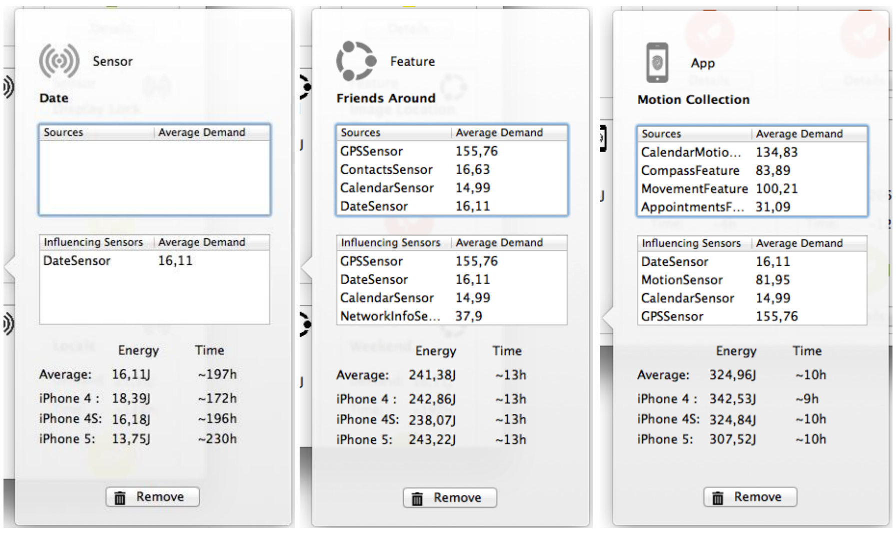
App
Motion Collection
Demand: 324,96J
Time: ~10h

App
Social Connector
Demand: 714,43J
Time: ~4h

App
Workload
Demand: 266,07J
Time: ~12h

Figure 2: Graphical user interface of the CONTEXTO Energy Demand Calculator

The previously introduced energy model serves as the basis for all estimations. In order to compute the energy demand of an application, the calculator searches recursively for related sensors, generates a unique sensor pool by removing duplicates. The construction kit of the calculator provides a *Sensor*, *Feature*, and *Application* component to interact with. This allows developers to rebuild their context-aware application and simulate possible configurations. The *Application* component represents the real-world application that is yet to be developed by holding a set of assigned feature components. A *feature* component is composed of necessary *Sensor* and *Feature* components, just as is the case in the framework. As lowest-level component, *Sensors* provide information about their average and device-specific energy demand. Based on this information, *Feature* and *Application* components accumulate their own energy demand, show the overall energy demand of their sources, and provide a breakdown of the energy demand of sensors they depend on. Figure 3 shows a detail views of all three component types. In addition, the calculator roughly estimates the time the component would need to completely drain a charged battery. This estimate helps developers to get a better sense of the actual impact of their application.


Sensor

Date

Sources	Average Demand

Influencing Sensors	Average Demand
DateSensor	16,11

	Energy	Time
Average:	16,11J	~197h
iPhone 4 :	18,39J	~172h
iPhone 4S:	16,18J	~196h
iPhone 5:	13,75J	~230h

Feature

Friends Around

Sources	Average Demand
GPSSensor	155,76
ContactsSensor	16,63
CalendarSensor	14,99
DateSensor	16,11

Influencing Sensors	Average Demand
GPSSensor	155,76
DateSensor	16,11
CalendarSensor	14,99
NetworkInfoSe...	37,9

	Energy	Time
Average:	241,38J	~13h
iPhone 4 :	242,86J	~13h
iPhone 4S:	238,07J	~13h
iPhone 5:	243,22J	~13h

App

Motion Collection

Sources	Average Demand
CalendarMotio...	134,83
CompassFeature	83,89
MovementFeature	100,21
AppointmentsF...	31,09

Influencing Sensors	Average Demand
DateSensor	16,11
MotionSensor	81,95
CalendarSensor	14,99
GPSSensor	155,76

	Energy	Time
Average:	324,96J	~10h
iPhone 4 :	342,53J	~9h
iPhone 4S:	324,84J	~10h
iPhone 5:	307,52J	~10h

Figure 3: Detail view in the CONTEXTO Energy Demand Calculator

For a still better orientation, the calculator introduces three general energy demand classes: *Low*, *Mid*, and *High*. Class membership is defined in terms of varying energy thresholds for sensors, features, and applications. Sensors belonging to the *low* energy class require barely more energy than an idle application. Sensors in the *mid* class have a noticeable impact, but are still far from the high energy demand class. The average energy demand of all sensors is approximately 40 J in 10 minutes. The classification uses this as a reference value and assigns sensors with an above-average energy demand into the *high* energy demand class. The remaining range is divided into two identical parts. Sensors belonging to the *low* energy demand class do not exceed an energy demand of 19 J. The *mid* class ranges from 20 J to 39 J. Table 2 provides an excerpt of the framework's sensor classification.

Low	Mid	High
BatterySensor	AccelerometerSensor	CameraSensor
CalendarSensor	AudioInSensor	GPSSensor
ContactsSensor	MuteSwitchSensor	GyroscopeSensor

Table 2: Energy demand classes for sensors (excerpt)

5. Related Work

The research presented here is embedded in a broad range of mobile, pervasive, and ubiquitous computing activities. Within these communities, there has been active research on context models, context recognition, and context-aware applications and devices. Concept and implementation of CONTEXTO highly benefit from this previous research. Various frameworks for context classification and recognition exist: *Context Toolkit* [8] supports developers in rapid prototyping of context-aware applications. The framework relies on a distributed architecture and provides sensor fusion. The *Hydrogen Context-Framework* [9] is based on a centralised architecture comprising adaptor, management, and application layers. The centralised design makes it more robust against network failures, and permits context use by multiple applications. *ContextDroid* [10] is an expression-based context framework that is implemented as an Android platform service. The framework utilises *context entities* that abstract from sensors. CONTEXTO is also related to research in the field of energy-aware software engineering and development: [11] presents the concept of energy labels for Android applications, a simple mechanism that easily allows end-users to assess the energy demand of their apps. *PowerTutor* [12] was one of the first energy models that were used for a mobile application. Numerous ongoing research regarding the measurement of energy demand on smartphones also exists (e.g., [3], [13]).

6. Future Work

CONTEXTO aims to provide energy awareness to developers of context-aware applications. In the future, we would like to make the framework itself aware of energy requirements. Using an energy budget system, developers will then specify a desired energy footprint, and the framework will make sure that the allocated budget is respected. This raises questions of the relation between energy demand and user requirements such as accuracy, precision, availability, or actuality of sensor data and context recognition. These parameters greatly influence the user acceptance of context-aware applications, which we will investigate further with the help of the framework. On a closer time horizon, we will conduct a user study with developers, to see if our design goals regarding the ease of use of the framework have been met.

References

[1] W. Clark, D. W. Cearley, and A. Litan. (2012, 8/5/2014). *Context-Aware Computing and Social Media Are Transforming the User Experience*. Available: http://www.gartner.com/doc/1916115/contextaware-computing-social-media-transforming

[2] V. Bellotti and A. Sellen, "Design for privacy in ubiquitous computing environments," in *ECSCW'93*, New York, NY, USA, 1993.

[3] H. Höpfner and M. Schirmer, "On Measuring Smartphones' Software Energy Requirements," in *ICSOFT 2012*, Rome, Italy, 2012.

[4] M. Schirmer and H. Höpfner, "Software-based Energy Requirement Measurement for Smartphones," in *First Workshop for the Development of Energy-aware Software (EEbS 2012)*, Braunschweig, Germany, 2012.

[5] M. Schirmer and H. Höpfner, "SenST: Approaches for Reducing the Energy Consumption of Smartphone-Based Context Recognition," in *CONTEXT'11*, Karlsruhe, Germany, 2011.

[6] M. Baldauf, S. Dustdar, and F. Rosenberg, "A survey on context-aware systems," *Int. J. Ad Hoc Ubiquitous Comput.*, vol. 2, 2006.

[7] S. Loke, "The Structure And Elements Of Context-Aware Pervasive Systems," in *Context-Aware Pervasive Systems: Architecture for a New Breed of Applications*, ed Boca Raton, FL, USA: Auerbach Publications, 2007, p. 25.

[8] A. K. Dey, G. D. Abowd, and D. Salber, "A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications," *Hum.-Comput. Interact.*, vol. 16, 2001.

[9] T. Hofer, W. Schwinger, M. Pichler, G. Leonhartsberger, J. Altmann, and W. Retschitzegger, "Context-awareness on mobile devices - the hydrogen approach," in *System Sciences 2003*, 2003.

[10] B. van Wissen, N. Palmer, R. Kemp, T. Kielmann, and H. Bal, "ContextDroid: an Expression-Based Context Framework for Android," in *PhoneSense 2010*, Zurich, Switzerland, 2010.

[11] C. Wilke, S. Richly, G. Püschel, C. Piechnick, S. Götz, and U. Aßmannn, "Energy Labels for Mobile Applications," in *First Workshop for the Development of Energy-aware Software (EEbS 2012)*, Braunschweig, Germany, 2012.

[12] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick, Z. M. Mao, *et al.*, "Accurate online power estimation and automatic battery behavior based power model generation for smartphones," in *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, Scottsdale, Arizona, USA, 2010.

[13] A. Pathak, Y. C. Hu, and M. Zhang, "Where is the energy spent inside my app?: fine grained energy accounting on smartphones with Eprof," in *Proceedings of the 7th ACM european conference on Computer Systems*, Bern, Switzerland, 2012.