Context Source: A Smartphone Application for Serving Context to a Generic Context Provisioning System

Michael Knappmeyer¹,², Daniel Brettschneider², Saad Liaquat Kiani¹, Ralf Tönjes²
¹University of the West of England, Bristol, UK
²University of Applied Sciences Osnabrück, Osnabrück, Germany

Abstract

This paper presents a modular context provisioning middleware for the support of ubiquitous services and applications, entitled C-ProMiSE (“Context Provisioning Middleware with Support for Evolving Awareness”). Special emphasis is put on a smartphone application which serves the middleware with primitive context originating from diverse physical, virtual and logical sensors (e.g. GPS, accelerometers, address book). Besides general design principles, a specific implementation based on the Android OS is prototyped and quantitatively evaluated. Since limited battery power is a major concern in smart mobile devices, the energy consumption by the prototype contextual application is also determined.

1 Introduction

Context-Awareness is an essential feature leveraging Ubiquitous Computing (ubicomp), the so called 3rd wave of mobile communication. In order to support users proactively during their everyday activities, their communication and interaction context is to be detected and made available to any context-aware application, service or actuator. Obviously, the specifically required context highly depends on the application domain and how context information is utilised for adaptation. Within the last decade of active research in the field of context-awareness, tourism, public transport, m-commerce, health and well being, e-learning, entertainment and gaming have been proposed as very popular and attractive usage scenarios. This variety and the fact that the interaction in ubiquitous systems results in a generic paradigm shift, encourage the hypothesis that there is no single killer application to be expected. Instead, context-aware systems have to support a huge diversity of existing and emerging applications simultaneously. Context information may comprise manifold data, e.g. time, environment, position, device capabilities, address, weather, activity, social relations, and intentions. Primitive context originating from diverse physical (embedded in smart phones, smart places or the environment), virtual (web services), or logical (information databases) sensors may be further processed to infer aggregated high-level context. Consequently, context comprises various abstraction layers and the need for gradual and diverse processing.

A well established practice for acquiring, processing and distributing context is the deployment of a ubiquitous middleware or a context provisioning system that hides the complexity underneath and offers a coherent access to context information, regardless of device and network access diversity. This paper describes C-ProMiSE, an extendible middleware for providing manifold context data to evolving and emerging applications. Special emphasis is put on a modular smartphone application serving the middleware with context data directly originating from the phone sensors. Smart phones are of particular interest since, being an everyday companion during everyday activities, they can offer essential insight into users’ situations. Furthermore, they are the primary personalised user interaction device with the digital world.

The remainder of this paper is structured as follows. Section 2 provides an overview of related work, before the C-ProMiSE middleware is introduced in Section 3. Section 4 presents the conceptual design of the Context Source application whose performance is further assessed in Section 5. Section 6 finally concludes the paper and summarises the lessons learnt.

2 Background and Related Work

Context-aware systems are designed to carry out a set of functions primary amongst which are data acquisition, context reasoning, context storage, context distribution and providing a platform for context-aware applications. A number of design approaches have been attempted for collective provisioning of these functions with the overall aim of making information available about anything, anytime, anywhere. Context-aware systems are usually designed as middlewares that adopt a layered design with each functional layer hiding the details of the underlying layers. The primary benefit of this approach is the encapsulation of varying complexities of different functions. Each layer builds on the information available to the layer below it e.g. context reasoning layer uses data collected at the data acquisition layer while the application platform layer inter-
acts with the context synthesis layer to retrieve context and does not concern itself with the details of data acquisition or synthesis process. Within this design approach, the provision of functions is often separated into different architectural components, e.g. some functions are provided by a central server while applications that use these functions are deployed remotely. The following paragraphs describe these common middleware designs.

**Direct Sensor Coupling** Usually adopted in early context-aware prototypes e.g. Cyberguide [1], this approach relies on directly accessing the sensors states on which the context information is based. Due to the tight coupling between the context system and the sensors, such systems are limited to simple use case and are often found in the form of context-aware applications on mobile devices. Generally, in such systems, a context-aware application directly consumes information retrieved from sensors and there are no dedicated context reasoning, communication or coordination functions.

**Context Server** In context server based designs, a central server performs the functions of collecting and synthesising context from sensor data and other information sources. Clients of the server access it remotely to access context or raw data to process locally. This distributed, client-server approach is useful for accommodating mobile clients by sharing the processing burden and sensing resources. This is one of the most common designs used in building context-aware systems. One of the reasons of prevalence of this design is the functional flexibility it affords to designers. For example, the central server may limit its functions to mere data acquisition or it may provide context synthesis and application platform on top of it. Prominent examples of this design include Guide [7], CoBrA [6] and CASS [11].

**Peer-to-Peer** In peer-to-peer design approaches, the functional tasks of context-awareness are carried out by peer components. It is a distributed design approach where peers either individually carry out mutually exclusive subsets of context-awareness related functions or replicate the functionality in different geographical or logical domains. These peer components usually utilise a centralised server for coordination of context with context consuming applications e.g. Hydrogen [16] and Context Toolkit [10]. Server-less peer-to-peer designs also exist and include ContextPeers [13], JCAF [3], CORTEX [22] but may suffer from limitations due to lack of a coordinating component e.g. the requirement of hard-coding addresses of communication endpoints between context services and clients in JCAF.

Different approaches to designing context-aware systems exist due to the constraints imposed by a variety of factors. Leading factors effecting the design decision include location of sensors, amount and mobility patterns on users, available resources, targeted scale of the system and methods of context acquisition and distribution. Direct sensor coupled design has been used when sensors and context consuming applications are accessible within a single system (e.g. a mobile device or a desktop computer) and there is not a pressing requirement for complex context reasoning. However, due to this tight coupling, the scale, scope and usability of these middlewares to be exploited as holistic context-aware systems is severely restricted. Most sensors have limited range and software components that access data from such sensors have to be located physically close to the sensors. These limitations triggered the evolution towards a context server approach where a central server acquires, processes and stores context while providing interfaces for local and remote applications to access context. Context server approach allows reuse of sensor data, relieves resource constrained devices from context acquisition and processing. While leveraging these benefits, context server approach requires consideration of new factors in the design that include selection of appropriate network protocols, quality of service parameters, network performance, mobility management etc.

The basic central server approach allows remote components to access context but the data acquisition function is still restricted by the limitation in range of sensors and other information sources. To overcome this shortcoming, the central server design has evolved further to allow distribution of the data acquisition function into multiple remote data acquisition modules that acquire data from their assigned sources and push it to the central server. This further distribution of functionality has transformed context-aware systems into truly distributed systems where each function may be hosted on different machines in a network and a central server coordinates the flow of information and control between these components. Other factors that have influenced the adoption of this design include availability of a large number of distributed information sources and sensors, dedicated reasoning components, mobility of modern day users and abundance and increased usage of mobile devices. Our context provisioning middleware, described in the Section 3, is built upon this central server design with distributed components. Other context-aware systems that have adopted this approach include Gaia [21], SOCAM [14] and PACE [15].

A recent change in the ecosystem of context-aware applications and systems is the availability of smart phones with an order of magnitude more capabilities than their counterparts that have been used in earlier context-aware systems. Factors including technological advancements, social trends and increased utility and adoption of mobile devices mean that context-awareness will be an important element in the next wave of the mobile-device centric services ecosystem. The problem is that the communication and computation of context on smart mobile devices will lead to considerably more energy consumption than that used up in current usage scenarios. This is plau-
sible because network communication and CPU processing in smart phones take up most of the energy resources under typical usage (around 80% on average according to (cp. [2], pg. 1989)). While there are a number of areas where energy conservation approaches can be incorporated (network transmission power, CPU scaling etc.), from the viewpoint of designers of context-aware systems it is imperative that the system design incorporates energy utilisation aspects of software performance. The computation power and memory (storage) capacity in mobile devices has continually increased - reducing in cost at the same time - but growth in battery capacity has not followed a similar trend. For example, the specific energy of lithium-ion battery commercialised in early 1990's holds only a small factor more of energy, approx. 100-250 Wh/kg (calculated from specifications provided at [20]), than much older lead-acid batteries (30-40 Wh/kg). Because of this limiting factor, even though modern mobile devices can do more, they can do so for only a limited time before the battery runs out of energy. The critical factor of limited energy in mobile devices requires optimised utilisation of resources in aspects of computation, display, interaction and communication. These optimisation mechanisms include on-demand CPU frequency scaling, turning displays off during calls, and variable transmission power for network communications. Based on these facts, our design of the Context Source application incorporates the awareness of energy consumption in its design.

Despite a number of context-aware systems in existence, energy consumption and conservation on mobile devices involved in context acquisition and utilisation has not been adequately addressed. Earlier systems were developed with the aim of demonstrating various functions of context-aware systems, e.g. context acquisition, management, representation, reasoning, and their prototype nature did not consider energy constraints. Moreover, the role of mobile devices in most context-aware systems has been limited to consuming context information in the form of a single executing application on the device. However, the role of mobile devices is becoming more central to our interaction with the digital world and their increased capabilities allow concurrent execution of a number of context consuming and producing applications and services. In the specific domain of context-aware systems, it is only recently that attention has been paid towards analysing the impact of context-aware applications on the energy consumption at the mobile device. Bernal et al. [4] have proposed a mechanism for reducing the context data publishing from mobile devices by adapting context publishing behaviour according to device conditions (signal strength, sensors status etc.). Similarly Kang et al. [17] emphasise that the major challenge in providing users with proactive services lies in continuously acquiring user and environment context from sensors due to the imposition of heavy workloads on mobile devices and energy consumption in performing this task. They attempt to address this challenge in their context-monitoring framework for sensor-rich and resource limited mobile environments, titled 'SeeMon'. The energy efficiency in SeeMon is based on optimal selection of the Essential Sensor Set (ESS) that can satisfy a context query. Crk, Bi and Gniady [8] propose a range of user-interaction-aware mechanisms that utilise an approach of monitoring a user’s interaction with applications through the capture and classification of mouse events to effect considerable improvements in energy savings and delay reductions of the WiFi network interface. Delvic, Graf and Barone [9] emphasise the need to retrieve context information from outside the device via network interfaces and evaluate the use of Bluetooth and WiFi interfaces for this purpose. They conclude multicasting over WiFi consumes less energy than Bluetooth and that it is more energy efficient to distribute context knowledge to other devices, than having each device learn or acquire this information itself. Zhuang, Kim and Singh [24] analyse the issue of energy efficient location sensing on smart phones and identify four critical factors that affect energy efficiency of location-sensing with GPS sensors. These factors include static use of location sensing mechanisms, absence of power-efficient sensors, lack of sensing cooperation among multiple location based applications and unawareness of the battery state. They present an adaptive location-sensing framework for Android-based smart phones, and evaluation results that show significant reduction in the GPS usage and improvement in battery life. The efforts discussed in these lines showcase the fairly recent trend recognising the importance of energy conservation in context-aware systems.

![Image](image_url)
3 Context Provisioning Middleware

In this section, the C-ProMiSE (Context Provisioning Middleware with Support for Evolving Awareness), a successor of the C-CAST system [18], is introduced. Figure 1 provides an overview of the middleware functionalities.

3.1 Context Modelling

Context is represented in C-ProMiSE according to the data model concept depicted below in Figure 2. For realising a light-weight representation schema suitable for processing on resource constraint mobile devices in a rapidly changing environment, an XML schema, entitled ContextML has been designed [19]. Most importantly, each context instance does not only comprise the context data itself (in simple parameters, arrays of similar parameters, and structures) but also context meta-information. The context meta-information contains the dynamically associated entity (unique identifier and type, e.g. user, device, smart room) and a specific scope. The scope can be interpreted as class object of a defined set of context parameters and type of context. Exemplar scopes are position, civilAddress, etc.

Each context instance is detected and exchanged on an atomic basis, i.e. in case of scope position, the parameters latitude and longitude can only be modified and exchanged at the same time. Furthermore, each context instance is only valid for a limited period of time, after which it becomes historic context.

3.2 Middleware Architecture

This subsection covers the middleware functionalities of Context Lookup & Discovery and Context Storage. The C-ProMiSE architectural concept for the management of context data applies a well known role model comprising consumers, sources, providers, and mediating brokers. All modules interact based on HTTP and ContextML [19] according to the REST (Representational State Transfer) [12] principle, and can be deployed across diverse devices and networks consequently. Each Context Provider (CxP) provides context information of a specific scope and is invoked synchronously. The internal processing logic depends on the desired output and may utilise any feature extraction or artificial intelligence mechanism. As input, sensor data may be fetched from physical sensors or external web services. Alternatively, the CxP may require context scopes originating from other providers as input. An internal database may be maintained if required. The output of a CxP is encoded in ContextML. Context Sources (CxS) asynchronously publish context directly to a CxB, either event-based or periodically. Each application or service querying for and utilising contextual information is a Context Consumer (CxC). A Context Broker (CxB) mediates between CxS/CxP and CxC modules to hide complexity and provide a single point of access. A CxB provides a CxB Registry & Lookup service, a Context Caching and a Context History service. Furthermore, it supports synchronous and asynchronous context request models and mechanism (cp. Section 3.4). It is important to highlight that each CxP announces its accessibility and capabilities in a ContextML encoded Advertisement message. As an optional feature, a CxP may announce its capability to dynamically translate between entity types (e.g. a User Profile CxP can associate a user to a device). In this way, new providers can easily be registered and be deployed during runtime while ensuring mobility and middleware evolution.

3.3 Context Processing

Since CxP components may rely on each other, context processing is performed in stages and in a distributed manner. Figure 3 illustrates the dependency between selected CxPs and their scopes, respectively. The directed edges imply that the successor node requires the ancestor as input. Exemplar output context parameters are summarised in Table 1. It can be easily inferred that primitive context (e.g. scopes Wifi and cell) plays a crucial role in detecting high-level context. This observation serves as main motivation for focussing on the design principles of a smartphone application providing such primitive context (cp. Section 4 and 5).
The design concept comprises not only the context collection performance of other applications. It allows for unobtrusive collection of context on the mobile design principles of our Context Source application which dresses book, interaction protocol). This section explains the functionalities required for collecting, processing and distributing the context are illustrated in Figure 4. The context collection service collects raw and device dependent data from a sensor. This data is saved in a local database for later use and is finally parsed into ContextML after having been processed (e.g. filtered and aggregated) optionally. The resulting context update message is sent to the C-ProMiSE middleware, more precisely to a CxB. Furthermore, the Context Source advertises its capabilities enabling the visualisation of the context data and providing configuration and privacy protection capabilities. The approach of the design is to send context asynchronously, either periodically in defined intervals or based on events. This way of asynchronous context exchange has been chosen due to the following three reasons: (1) The context can be instantly accessed from the C-ProMiSE middleware without having to invoke the smartphone synchronously which would result in significant delay. (2) Asynchronous system-initiated actuation of communication services is facilitated. The system is instantly up-to-date about user related information. (3) Accessing smart phones synchronously is often problematic due to loss of connectivity, mobility, port restrictions or firewalls. The context is separated into context scopes. The context data is retrieved from physical or virtual sensors as summarised in Table 2.

### 3.4 Context Distribution

For efficient distribution of contextual information, three context query models are supported by C-ProMiSE: (1) A simple on-demand query model allows a CxC to define the desired triple of entityID, entityType, scope and synchronously receive the current context (or a negative acknowledgment message if the context is not available). In this case a CxB provides a proxy query mechanism, i.e. it fetches the required input context consecutively from CxPs if not present in cache already and requests entity translation. Therefore, the query is accelerated and simplified from the viewing angle of a resource constraint CxC. (2) By utilising ContextQL [19], an XML schema for defining constraint context queries, a CxC can synchronously query context of more than one entity matching specific constraints. For instance, context of all user entities being located at a certain location (civilAddress.city = Bristol) can be queried. Several logical and comparison operators are supported as further detailed in [19]. (3) The ContextQL can also be used by a CxC to define context changes of interest. An asynchronous publish/subscribe mechanisms allows event-based notifications to be sent.

### 4 Context Source Design

In order to support a broad variety of context to be inferred by the C-ProMiSE middleware, primitive context about the users’ interaction, their devices and their activities has to be provided to the middleware. In their role of personalised everyday companions, smart phones can provide diverse information originating from physical sensors (e.g. GPS, accelerometers) and virtual sensors (e.g. address book, interaction protocol). This section explains the design principles of our Context Source application which allows for unobtrusive collection of context on the mobile phone without disturbing the user and influencing the performance of other applications.

The design concept comprises not only the context collection itself but also a GUI (Graphical User Interface) enabling the visualisation of the context data and providing configuration and privacy protection capabilities. The approach of the design is to send context asynchronously, either periodically in defined intervals or based on events. This way of asynchronous context exchange has been chosen due to the following three reasons: (1) The context can be instantly accessed from the C-ProMiSE middleware without having to invoke the smartphone synchronously which would result in significant delay. (2) Asynchronous system-initiated actuation of communication services is facilitated. The system is instantly up-to-date about user related information. (3) Accessing smart phones synchronously is often problematic due to loss of connectivity, mobility, port restrictions or firewalls. The context is separated into context scopes. The context data is retrieved from physical or virtual sensors as summarised in Table 2.

### Table 1: Scope Parameters and CxP Processing

<table>
<thead>
<tr>
<th>Scope</th>
<th>Context Parameters</th>
<th>Source and/or Processing</th>
</tr>
</thead>
<tbody>
<tr>
<td>userProfile</td>
<td>firstname, lastname, gender, birthday, messengers, ... address, imei, phoneNumber, emailAddresses, ...</td>
<td>database lookup (internal)</td>
</tr>
<tr>
<td>wf</td>
<td>wiList, wiRsolList, wiDevices, ...</td>
<td>Android API</td>
</tr>
<tr>
<td>cell</td>
<td>cellList, cellRsolList</td>
<td>Android API</td>
</tr>
<tr>
<td>motion</td>
<td>accSum, accX, accY, accZ, accDelta</td>
<td>Android API</td>
</tr>
<tr>
<td>campuslocation</td>
<td>room, building, accuracy</td>
<td>Naive Bayesian Classifier</td>
</tr>
<tr>
<td>position</td>
<td>latitude, longitude, accuracy</td>
<td>Google GEARS REST API</td>
</tr>
<tr>
<td>time</td>
<td>timezone, localTime, localDate, localDayOfWeek, daytime, noon, afternoon, ... weekend, season</td>
<td>GeoNames REST query for Timezone; Fuzzy logic for interpretation</td>
</tr>
<tr>
<td>civilAddress</td>
<td>street, zipCode, city, country, accuracy</td>
<td>Google Maps REST API</td>
</tr>
<tr>
<td>place</td>
<td>mostRelevantPlace.label, mostRelevantPlaceCategory, relevance values per category and label</td>
<td>User-Feedback based geographic database, weighted accumulation</td>
</tr>
<tr>
<td>weather</td>
<td>currentWeather summary, temperature, humidity, windSpeed, windDirection, forecastedWeather, ...</td>
<td>Web Service Query</td>
</tr>
<tr>
<td>activity</td>
<td>shopping, working_in_office, working_at_home, eating, sleeping</td>
<td>Bayesian Network Inference</td>
</tr>
</tbody>
</table>
Table 2: Scopes Served by Context Source

<table>
<thead>
<tr>
<th>Scope</th>
<th>Description</th>
<th>Sensor</th>
<th>Default Interval</th>
<th>Update</th>
</tr>
</thead>
<tbody>
<tr>
<td>cell</td>
<td>connected and neighbouring cell towers with RSSI (Radio Signal Strength Indicators)</td>
<td>TelephonyService</td>
<td>5 Min.</td>
<td></td>
</tr>
<tr>
<td>Wifi</td>
<td>connected and neighbouring Wifi routers with RSSI</td>
<td>WifiService</td>
<td>2 Min.</td>
<td></td>
</tr>
<tr>
<td>bluetooth</td>
<td>connected and neighbouring Bluetooth devices with RSSI</td>
<td>BluetoothService</td>
<td>10 Min.</td>
<td></td>
</tr>
<tr>
<td>motion</td>
<td>accSum, accX, accY, accZ, accDelta</td>
<td>Accelerometers</td>
<td>1 Min.</td>
<td></td>
</tr>
<tr>
<td>deviceSettings</td>
<td>basic settings of device</td>
<td>TelephonyService, AudioService, WindowManager</td>
<td>10 Min.</td>
<td></td>
</tr>
<tr>
<td>deviceStatus</td>
<td>status information of device</td>
<td>TelephonyService, WifiService, BluetoothService, ConnectivityService</td>
<td>10 Min.</td>
<td></td>
</tr>
<tr>
<td>tasksInfo</td>
<td>running tasks</td>
<td>TaskService</td>
<td>10 Min.</td>
<td></td>
</tr>
</tbody>
</table>

Figure 4: Context Source conceptual design layers

started. Furthermore, appropriate locks ensure its functionality even if the phone is in sleep or standby mode. The phone is waken up when a service has to collect data and prevented from going back into energy saving mode for the entire duration of collection and processing of measurements.

For interaction with the C-ProMiSE middleware, validity duration is defined (cp. section 3.1) and the collected data has to be parsed into ContextML. For enabling the user to monitor the information which is sent to the system, each context instance is saved in a local database and presented human readable in the GUI if the user opens the application. The ContextML message is then sent to a Context Broker as a context update message and stored in its cache. The update intervals (in case of periodic updates) can be either configured by the user in the GUI or adapted remotely. The remote configuration also allows for definition of events, i.e. the Context Source sends an update only if the defined rules match. To allow for dynamic reconfiguration capabilities the Context Source advertises itself to a C-ProMiSE CxB. Figure 5 depicts the corresponding communication flow. The Context Source parses the possible settings of each scope, i.e. activation (on, off), report-Interval (in minutes), eventMode (periodic, event based), and eventDefinition into a ContextML message and advertises itself with this message at a C-ProMiSE CxB along with basic contact information of the phone which consist of the current version of the Context Source and the protocol and id where to find it. In this example, MQTT (MQ Telemetry Transport) is used as message push protocol. With this information the broker can send remote configuration to the phone which can dynamically adjust the functionality of the Context Source if permitted by the user settings. In order to allow for modular extendibility of the Context Source every service implements a well-defined interface. If a new phone sensor is to be supported, a new service can easily be build in order to integrate it into the Context Source. To ensure privacy for the user there is a simple way to stop the Context Source or to enable or disable selected scopes.

5 Evaluation

5.1 Prototype Implementation

The implementation of the Android Context Source differs marginally from the initial designed discussed in the preceding section. The modifications in the design, which is shown in Figure 6, arise due to some characteristics of the Android API.
The design is extended by a Status Service that controls the context collection sub-services as illustrated in Figure 7. This Status Service is woken up by an Alarm Manager of the Android API which functions as a wakeup timer. It also implements the queue where the different context scopes are scheduled for their execution. When the Status Service is periodically triggered it checks the built-in queue for upcoming context collections and invokes the corresponding Scope Service. When there is no scope to schedule the Context Source application goes back into standby. When a particular ScopeService gets started it uses the system service of the Android API to access the sensor which is represented by the scope. The raw sensor data is saved in the database to allow displaying the collected data to the user when required. The data is then parsed into ContextML and sent to the C-ProMiSE system as a context update message. The ScopeService reports a success to the StatusService which reschedules the scope according to its interval. When the collection of the sensor data or the transmission fails, an error message is reported to the StatusService that invokes a back off procedure to ensure availability of context.

The communication between the services (cp. Figure 8) is mainly realised through broadcasts. Asynchronous broadcasts are system wide messages which can be received by a broadcast receiver. Since the application has a timeout limit of fifteen seconds all heavy tasks have to run in their own thread. For that Android offers asynchronous tasks (AsyncTask). The scope services invoke these tasks to avoid blocking the GUI especially during HTTP accesses. Figure 9 shows selected screenshots of the implemented application.

5.2 Experiment Setup

The power consumption is measured with an application called PowerTutor [23], which is an application for Google phones that displays the power consumed by major system components such as CPU, network interfaces, display, etc. and different applications. This application allows software developers to see the impact of design changes on power efficiency. PowerTutor calculates the phone’s breakdown of power usage with an average of 1% error over 10 second intervals while the worst case error over 10 seconds is 2.5% [23] (pg. 8).

The experiments are carried out on an HTC Desire phone with the Android platform version 2.2. Five measurements are taken for every experiment with durations of one hour each. The main focus of these experiments is to compare the power consumption of the context scopes acquisition, processing and updates during this time, rather than to obtain absolute measurements. A single non-moving device was used with a connectivity of 75-100% to achieve these results.

5.3 Evaluation Results

The first set of experiments is performed with a WiFi connection. Figure 10a illustrates the average power consumption measured with different context scopes and varying update interval. When the update interval is 10 minutes or longer, the acquisition, processing and update of all scopes consume approximately 4mW of power. This update interval of 10 minutes results in low usage and hence is quite energy saving but is not suitable for fast changing context e.g. motion context acquired through physical sensors (accelerometer, gyroscope etc.). Lowering the context update interval to 5 minutes does not have a big impact on the energy consumption and costs nearly 1.5mW. Changing the interval to 2 or 1 minutes results in a bigger impact on the battery and the power consumption doubles as compared
Figure 9: Snapshots of various screens of the Context Source application

Figure 10: Experiment results
to the previous case. Overall the scopes that access physical sensors have a bigger impact on the power consumption when compared to virtual sensors.

In the next set of experiments the distribution of power consumption in provisioning of the three context scopes is measured. The update interval of the Context Source application is set to 1 minute for all scopes. The first measurement is the ScopeService without acquiring or updating the scopes i.e. idle execution. The power consumption by the service in this state is about 4mW. The next task is to collect sensor data and then to parse the collected sensor data into ContextML and the finally send this ContextML as a Context Update to the C-ProMiSE CxB. Cell and Wifi scope differ only slightly in energy consumed during sensor measurement and the main difference between the two is in the length of the ContextML data they process. The sensor collection takes up 0.2 to 0.4 mW of power but the ContextML document size is different (Wifi 1763 bytes with 6 neighbours and cell 711 bytes with 6 neighbours) which also affects the power consumption during sending of the respective context updates. The biggest impact is in the use of a physical sensor like motion (accelerometer and gyroscope) which has a similar data length to the ContextML data of the cell scope (751 bytes) but the collection of the sensor data consumes nearly as much power as the whole procedure with cell or wifi. The collective and relative impact of context acquisition, processing and update of these scopes is shown in Figure 10b.

6 Conclusion

In this paper we have presented an extendible middleware for the provisioning of diverse context information, entitled C-ProMiSE. We have argued the importance of primitive context originating from smart phones and presented the design of a Context Source application which has been implemented based on the Android SDK. A prototype demonstration of the Context Source application that acquires, processes and submits various primitive context scopes to the C-ProMiSE middleware has also been discussed along with various aspects of energy consumption on the smartphone.

The presented measurements have been taken in a lab prototype. Due to sufficient memory and processor power on the selected device, distractive influences on other applications running on the device have not been perceived. Hence, the evaluation concentrates on assessing the energy consumption which is especially important if sensor readings, context processing, and context exchange are constantly performed in the background. The required energy consumption has been determined with varying sensors and while utilising different radio access networks. A realistic setting with real hardware has been chosen, hence the results are useful for being taken into consideration while developing a context management/provisioning middleware on top of smartphone sensors.

The Context Source has been uploaded to the Google Market [5] and is freely available for public use. We aim at evaluating the C-ProMiSE middleware in-situ in a large-scale field study. A Group Catcher application has already been developed for utilising manifold context and clustering individual users dynamically. It allows for party planning, sharing, and other community-based entertainment communication. In addition, we are currently in the progress of implementing a Campus Assistant to be used in university campus environments for context-aware services. Both applications embed tailored advertising by asynchronous system-initiated message pushing.

7 Acknowledgments

Parts of this work have been conducted in the research project “XonYt” being funded by the German Federal Ministry of Economics and Technology.

References


