Developing a Location-aware Calendar App Using a Mobile Context Platform

Jörg Roth

Univ. of Appl. Sciences Nuremberg
Kesslerplatz 12
D 90489 Nuremberg, Germany
Joerg.Roth@Ohm-hochschule.de

Abstract: In this paper we present a location-aware calendar app that fully runs on a mobile device without the need for a central service. Calendar entries that are more related to the current context are displayed more obtrusively, whereas other entries are presented unobtrusive or even hidden. This allows a user to get an optimal overview about the entries that are relevant for the current situation. In addition, the calendar tool only emits sound reminders, if they are suitable for the current context. The tool is developed on top of the context-provider *Zonezz*.

1. Introduction

If a mobile application knows about the context, it can provide more specific information suitable for the current situation. This is especially important as mobile devices have certain limitations that affect the usability for mobile users. In this paper we present a new context-aware calendar app *DateStoneZZ*, which is an extension of the existing app *DateStone Calendar* [Ro11a].

The DateStone Calendar is a typical calendar comparable to hundreds of similar apps. It allows a user to enter and view tasks and dates and to switch between different views (e.g. day or month). The user classifies entries with the help of user-defined categories (e.g. 'Home', 'Job', or 'Sports'). As a special feature, DateStone additionally can *speak* reminder alarms using text-to-speech-service.

Our goal was to extend this calendar to react on contexts. As a major requirement, the calendar should fully run on a mobile device without the need for a central service. This is important as context information and locations are considered as private data.

We identified three calendar functions that should be affected by the current context. DateStoneZZ is built on the context-provider Zonezz – a context-platform that can be used by different apps on a mobile device. Both DateStoneZZ and Zonezz are fully implemented for the Android platform.

2. Related Work

Tools that react on contexts and context changes were already presented years ago. In [MS00], e.g., reminders are generated automatically when a user enters a certain location. A tool can e.g. produce a reminder alert to buy something when a user enters a shopping site. Even though the basic ideas are old, the resulting platforms were not suitable for typical end-users as equipment and software were highly specialized.

One idea to deal with contexts is to identify *symbolic* locations. In [Ve09] three major contexts are distinguished: 'home', 'work' and 'on the move'. [AS03] presents an approach to geometrically cluster GPS locations of multiple users to identify interesting locations. Also indoor locations are considered: in [KTD+03] small indoor regions, so called *activity zones* (e.g. the 'lunge chair') are classified. Activity zones may have complex geometric shapes and can trigger context rules that can perform further activities. Also non-traditional means of positioning are considered: in [NCGG10], apps can distinguish types of movement (such as 'walking', 'car' and 'train') with the help of the accelerator sensor that is available in most smart phones. The movement type can be viewed as additional information to define a context.

Projects such as *CybreMinder* [DA00], *Nexus* [HHL+10] or [SHW+06] provide a general platform to capture and distribute context information. Usually they base on a centralized infrastructure that collects information and executes the rules to trigger events. An exception is *ContextPhone* [ROPT05] that primarily provides a system to plug-in context components on a mobile device. If a platform relies on a central service, privacy issues become more important.

Even though conceptual interesting, some of the approaches above provide too many degrees of freedom to define contexts and rules. A typical user wants to understand such a system in some degree. If we want to avoid security issues, it is reasonable to keep the entire computation on the mobile device.

3. A Context-aware Calendar

The context-awareness of DateStoneZZ is based on two concepts: *categories* and *zones*.

Categories provide a classification of date entries defined by the user. Typical categories are 'Home', 'Hobby', 'Travel', or 'Meeting'. A user can specify an arbitrary number of categories that are (in the non-context case) primarily used as display filters. DateStone categories primarily specify a name and a colored icon.

The second concept is the *zone*. A zone represents a *meaningful* area, *where* a user can reside, e.g. 'Home' or 'Job'. The zone is primarily represented by its name. The current zone is computed by the context-provider *Zonezz* (see section 4), i.e. the calendar has not to deal with context detection.

Zones and categories are strongly related and have some overlap. For many entries, a category is only important for a certain zone, e.g. a category 'Meeting' usually occurs in the Zone 'Job'. We thus basically use a matrix of *category* × *zone* (we call *zone rules*) to control context-dependent calendar features.

3.1 Controlling Calendar Features

We identified three calendar features that are affected by the current zone:

1. The color style is changed according to the current zone. There can be e.g., a different 'work' and 'home' color style. The look and feel of app usage is heavily influenced by colors, thus it is useful to have different color styles for different contexts. There exist predefined styles such as 'black/white', 'water', 'summer' that define all screen colors.

2. Date entries that are more related to the current context are displayed more obtrusively (e.g. bold) whereas other entries are presented unobtrusive or even hidden. E.g., job-related tasks are recognized very fast at a single glance when at work, whereas hobby tasks are not bothering.

3. Reminder alarms may be delayed until the user enters a certain location. E.g. an alarm to remind something special to buy in the supermarket may be delayed until the user enters a shopping site. The current zone should affect both the notification sound and the spoken text generated by the speech-to-text service.

Fig. 1 shows the same day sheet displayed in different zones. In the 'Work' zone (left), work entries are bold, others are transparent or even hidden. In the 'Home' zone (middle), home entries use a standard font, whereas work entries are transparent. In the zone 'Shopping' (right) only things to buy are bold.

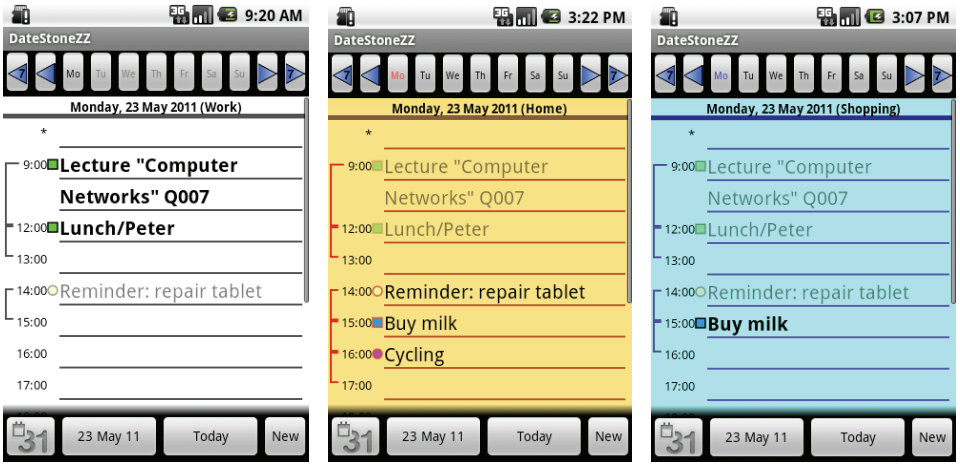

Figure 1. Different presentations dependent on the current zone

A user can express the relevance of a category for a certain zone with the help of text styles:

- *Hide*: the entry is not shown at all. It does not even occupy space in the time line of the calendar sheet.
- *Transparent*: the entry is shown with less density.
- *Show*: the entry is shown with normal density.
- *Bold*: the entry is shown with bold font.

To reduce the matrix of *category* × *zone*, the user does not have to edit all zones but can use the tokens 'inside' and 'outside'. The latter usually covers multiple zones and significantly reduces the effort to administrate the zone rules. As a result, we get a matrix as presented in table 1.

Table 1. Example zone rules to control the calendar presentation

			Calendar Category			
		Color Style	**University**	**Home**	**Hobby**	**Shopping**
Rule	Inside 'Home'	Summer	Transparent	Show	Show	Show
	Inside 'Work'	B/W inv.	Bold	Transparent	Hide	Hide
	Inside 'Shopping'	Water	Transparent	Transparent	Hide	Bold
	Outside 'Home'	Desert	Show	Transparent	Show	Show

Besides the visual representation, controlling acoustic alarms is an important issue. This is especially true when calendar entries are spoken such as '*Meeting with Peter in two hours*'. As calendar entries often are very private, they should not always be spoken. For this, we extend the zone rules and get an additional matrix as presented in table 2. For each combination we can define one of:

- *Speech+Alarm*: an alarm sound is emitted and the text of the calendar entry is spoken using the text-to-speech service.
- *Alarm*: only the alarm sound is emitted and it is not spoken.
- *Delay*: no sound is produced, but the alarm is stored for a later notification when the user enters a zone that permits this alarm.

The user can define each of the rules above for alarms in the next day ('1' in table 2) and later alarms ('L' in table 2).

Table 2. Example zone rules to control alarms

		Calendar Category			
		University	**Home**	**Hobby**	**Shopping**
Rule	Inside 'Home'	1,L:Alarm+Speech	1,L:Alarm+Speech	1,L:Alarm+Speech	1:Alarm+Speech L: Delay
	Inside 'Work'	1:Alarm+Speech L: Alarm	1,L: Alarm	1,L: Delay	1: Alarm L: Delay
	Inside 'Shopping'	1: Alarm 1: Delay	1: Alarm 1: Delay	1,L: Delay	1,L: Alarm
	Outside 'Home'	1: Alarm L: Delay	1: Alarm L: Delay	1,L: Delay	1,L: Delay

One could argue that the definition of these matrixes could be demanding for a typical user. However, the *Zone Rule Editor* inside the calendar tool simplifies to enter the rules as useful settings are pre-defined.

3.2 The DateStoneZZ Architecture

The architecture of the calendar app is presented in fig. 2. The left part of the architecture shows the old, non-context-aware application. The most interesting components are the *Calendar Sheet Renderer* that creates a date view, and the *Alarm Generation* that is responsible to create notification alarms. More components exist, but are not related to the context-aware extension, thus not presented for clearness. All components base on persistent storage, mainly SQLite databases and app preferences.

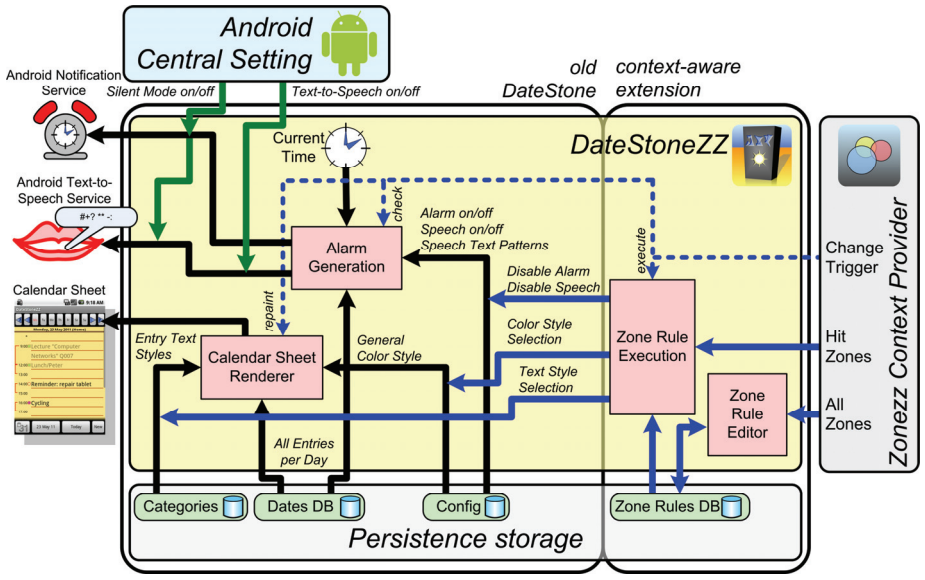

Figure 2. Data flow in the context-aware calendar

The major context-aware components are the *Zone Rule Editor* and the *Zone Rule Execution*. The latter reacts on changes of the current zones and controls the rendering and alarm generation. They both base on the *Zone Rules DB* that stores the zone rules.

The important information about the current zones is provided by the Zonezz context-provider presented in the next section.

4. The Context-Provider

It is reasonable to shift administration and detection of contexts to a platform on a mobile device rather than separately inside each app. All context-aware apps can use such a platform with the help of powerful communication and triggering mechanisms between software components that are available in modern smart phone operating systems. Our context-aware calendar is based on the Zonezz platform that provides central location-based context detection on a smart phone. Zonezz is designed according to the following goals:

1. The entire context detection is performed on the mobile device. No central service or central database is required.

2. The concept is understandable and manageable by typical users. Complex context configurations or rules are avoided.

3. The user is strongly supported by the platform to set up the configuration, but is able to overwrite platform settings if desired. Zonezz contains a suggestion system that makes recommendations for new important locations based on the position history.

We do not want to distinguish fine-grained contexts such as 'meeting in room xy' that require a very precise indoor positioning system. Typically we want to distinguish 2 to 4 contexts of which 'work' and 'home' usually are the most important.

Our approach is not designed for security related applications, e.g. for access control or payment. Position measurements can easily be manipulated inside the phone and the resulting contexts are thus not verifiable. Even tough there exist complex mechanisms to generate manipulation-secure positions [De09], they are not incorporated into our platform.

All context-relevant data are stored on the mobile device. As the whole computation is also performed on the smart phone, we have not to deal with privacy issues related to the network or central services. No information from our platform is sent via a wireless network. But, if attackers get access to the mobile device itself (e.g. steal it or get a root login), they may be able read private context-related data of our platform. These attacks, however, would be a serious problem for nearly all mobile applications.

The Zonezz core system has only a total size of 71 kB thus even suitable for less powerful smart phones.

4.1 Defining Zones

Our model is influenced by former research (especially [Ro08]) but is strongly simplified as the entire execution should take place on the mobile device. We use a simple geometric model to define different contexts called *zones*. Zones are two-dimensional circular regions on the Earth's surface with arbitrary center and radius. The model is easy to understand by the mobile user and allows the execution of efficient algorithms. To define a new zone, a user points on a map and specifies name, color and radius (fig. 3 left, middle). The color is only used for presentation purposes. For radiuses, there exist discrete values 50m, 100m, 200m,…, 50km.

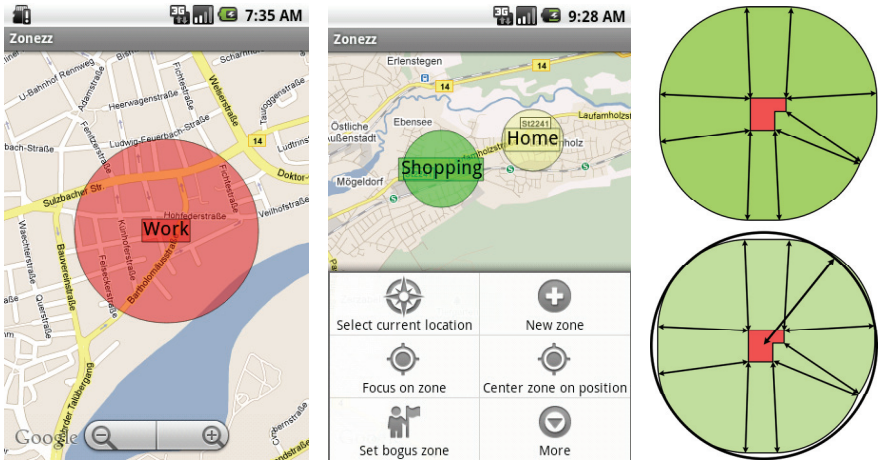

Figure 3. Zones

One could argue that circles usually do not precisely represent context borders. But:

- Usually, these shapes are sufficient to separate contexts by their area.
- More complex areas such as polygons are difficult to enter and to administrate on a mobile device.
- For circular areas we can realize efficient algorithms to suggest new zones.
- Position measurements are Gaussian distributed around a real position (fig. 3 right). Areas that include all position measurements of a certain area tend to circles anyway, even though the original areas are not circles. Thus, the actual benefit for more complex geometries would be small.

Note that zones very often mark an indoor-environment, thus GPS cannot be used. For this, Zonezz activates the 'network' position provider available on many devices. This positioning system measures WLAN and mobile phone signal strength fingerprints. The precision is low compared to GPS and measured position may deviate some hundred meters from the origin, thus we get the typical area of measurements as presented in fig. 3 (right).

Every time a measured position is inside such a circle, the corresponding zone is considered as part of the current context. As zones may overlap, the context is defined by a set of *hit zones*. We consider the hit zone with the smallest radius as more specific for the context, thus the set of hit zones is ordered ascending by their radius.

4.2. The Zonezz Architecture

Fig. 4 shows the Zonezz architecture. The platform contains the *Zonezz App* and *Core Services* with a total size of 71 kB. The app is mainly used to set up the zone database (*Zone DB*), which is modeled with SQLite and contains the table of zones. New zones can be suggested by a *Suggestion System*.

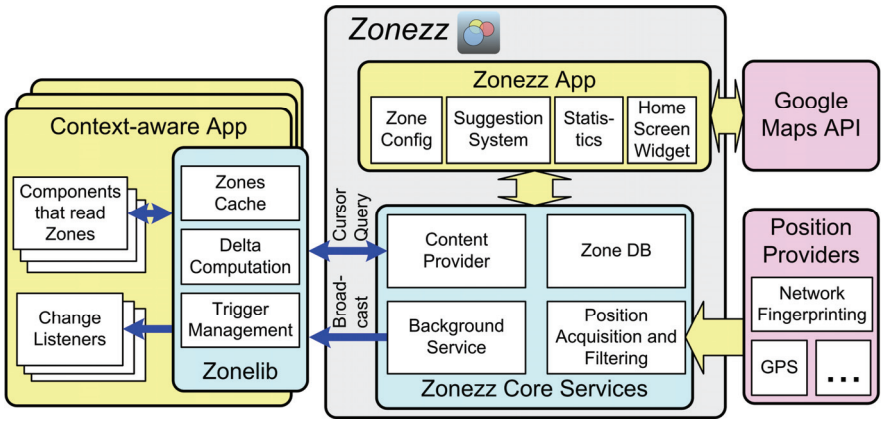

Figure 4. The Zonezz architecture

The Android structure to query data across apps is the *Content Provider*. Similar to an SQLite database the Zonezz content provider supports cursor-based queries on the tables *all zones*, *hit zones* and a *state*.

The *Zonezz Background Service* permanently computes the current set of hit zones. For every change it sends a system-wide broadcast using Android's *Broadcast Receiver* model. Any app may register for these events to asynchronously react to changes. The check for zone changes does not significantly affect battery lifetime. Note that on typical smart phones a lot of background services periodically work.

To compute the list of hit zones, the recent position is required. Current Android devices offer two providers: GPS satellite navigation and a 'network' provider that is based on signal strength fingerprinting. As a main decision, a user has to specify, if GPS should be turned on (causing higher battery consumption). The core component *Position Acquisition and Filtering* tries to fulfill the user's and system's requirements according precision, age of measurements and battery consumption. It balances these factors with the help of a set of pre-defined rules.

Here are some rules that the components considers:

- If the user activates GPS in the Androids central setting, Zonezz tries to access GPS.
- If the last GPS position is too old, the network position is used instead (if allowed by the user).
- If a last position fix is too old, the last position is considered as 'no position'. This leads to an empty hit zone list.
- To keep the GPS mechanism running, Zonezz periodically activates GPS. It turned out that simply listening to new GPS updates is not sufficient.

An empty hit zone list can be result of two effects: either no position measurement is available or a position is available but outside of all zones. An application maybe has to distinguish these cases. Thus, the Zonezz content provider and broadcasting components distribute *state* information that indicates, whether the last measurement was successful.

To simplify the access of other apps to Zonezz services, they can use the *Zonelib*. This library shields the communication, provides caching and manages the registration of listeners. Zonelib has a very small size of only 6 kB (jar file) and can simply be included into any Android app project.

4.3 Recommending New Zones

Zonezz contains a *Suggestion System* that makes suggestions for new zones based on the position history. The idea is to identify clusters of measurements that are close together.

An ideal Suggestion System would compute new zones without the help of the user. However, it is not clear for the system, whether a user really moves in an area or if the measured positions deviate due to measurements errors. In addition, the system does not know, if a cluster of positions has to be divided into multiple zones because some parts have different meanings to the user. Therefore, we need user cooperation. The user defines the time range of position measurements that should be analyzed (e.g. last day, last week), and the minimum time a user has to reside at a location to consider it as a new zone (e.g. 4 hours/day). A user who, e.g., requests a suggestion for a new zone 'work' could demand a minimum of 7 hours per day in the last three days.

The idea of the suggestion algorithm is as follows:

- Each measurement in the specified time range is considered as center of a potential zone. We iterate through all measurements and all possible radius steps. Note that our radiuses have discrete values. For every iteration we count the number of other measurements that are inside the circle.
- We filter out iterations that do not fulfill the minimal residence time defined by the user.

- For each iteration we compute a rank that reflects the number of measurements (more is better) and the radius (smaller is better). We filter out zones below a certain rank.

- Remaining iterations are mapped to zone suggestions. As the suggested center we use the mean of measurements inside the circle and *not* the measurement that originally defined the circle.

We order the remaining zones by their rank (best first). Each zone inside the list is presented on a map. The user is asked one after the other, whether the suggestion should be stored as new zone or whether the next suggestion should be presented (fig. 5).

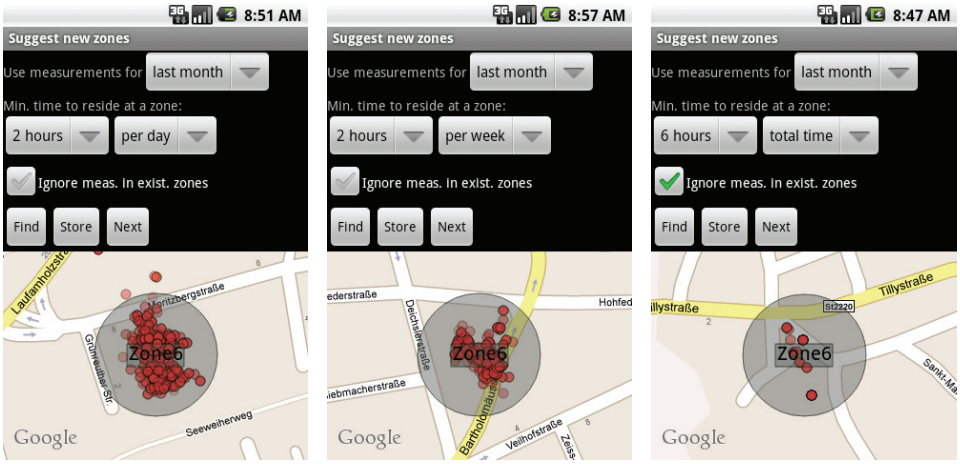

Figure 5. Suggesting zones

We could use very different formulas to rank the potential zones. We made the best experiences with $rank = m / \sqrt{r}$ for m measurements inside the circle of radius r.

Some words about performance. Zonezz logs a position measurement every 6 minutes. If a suggestion should base on one month, this means to check 7200 records. For n records a plain implementation requires $O(n^2)$ steps to compute all ranks. If we used spatial indexing mechanism such as [Ro11b], we can reduce the number of checks to $O(n \cdot \log n)$. Zonezz requires 21 seconds to create suggestions based on one month of measurements on a Motorola Milestone device (550 MHz).

4.4 Further Functions

The Zonezz platform offers further functions:

- The user can put a widget onto the Android home screen (fig. 6 left). This is useful to view the current zone and positioning state.

- A position statistics (fig. 6 middle) provides information, how long the user resides at which zone.
- The user can get a position history of visited places (fig. 6 right).
- The user can set a 'bogus zone' for a certain time. This means, the measured positions are ignored and a pre-selected zone is returned as hit zone.

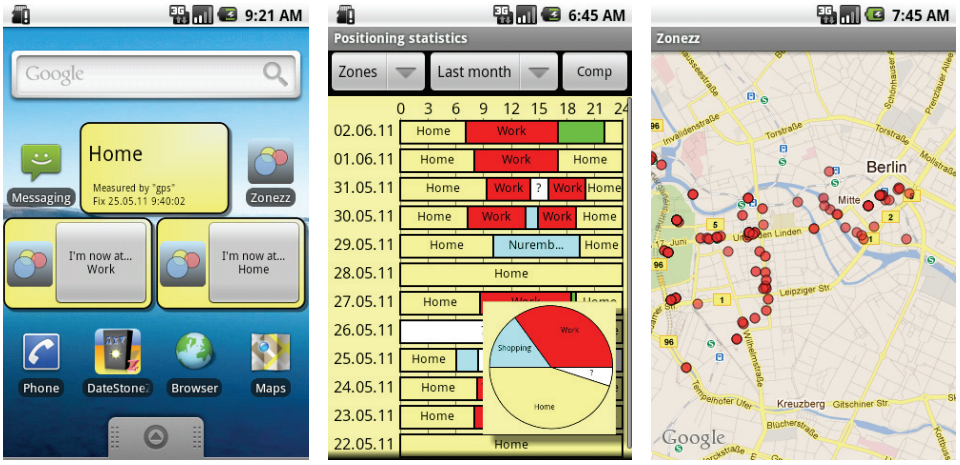

Figure 6. Additional functions

To set a bogus zone, the user can put a home screen widget (fig. 6 left) that can be controlled with one single click. Bogus zones can be useful if, e.g., real positioning completely fails in a certain situation, or if the user wants to simulate a certain context.

5. Conclusions and Future Work

DateStoneZZ is a significant improvement of a traditional calendar application. It only presents useful information for a specific context and the user can read information much quicker. In addition the current context controls, whether sound alarms and spoken reminders should be emitted.

In the future we want to additionally integrate movement into the concept. Besides our zones, contexts such as 'train-driving', 'walking' etc. could be derived from the accelerator sensor and further confine the user's current situation.

References

[AS03] Ashbrook D., Starner T. 2003. Using GPS to Learn Significant Locations and Predict Movement across Multiple Users. Personal and Ubiquitous Computing archive Vol. 7 No. 5, Oct. 2003

[De09] Decker M. 2009. Prevention of Location-Spoofing – A Survey on Different Methods to Prevent the Manipulation of Locating-Technologies. Proc. of intern. conf. on e-Business (ICE-B 09), Milan, Italy, July 7-10 2009, 109-114

[DA00] Dey A., Abowd G. 2000. CybreMinder: A Context-Aware System for Supporting Reminders. Proc. of the Handheld and Ubiquitous Computing 2000, Bristol, UK, Springer LNCS 1927, 172-186

[HHL+10] Häussermann K., Hubig C., Levi P., Leymann F., Simoneit O., Wieland M., Zweigle O. 2010. Understanding and designing situation-aware mobile and ubiquitous computing systems. Proc. of intern. Conf. on Mobile, Ubiquitous and Pervasive Computing, March 2010, 329-339

[KTD+03] Koile K., Tollmar K., Demirdjian D., Shrobe H., Darrell T. 2003. Activity Zones for Context-Aware Computing. UbiComp 2003: Ubiquitous Computing, Springer LNCS 2003, Vol. 2864/2003, 90-106

[MS00] Marmass N., Schmandt C.: Location-Aware Information Delivery with ComMotion. Proc. of the Handheld and Ubiquitous Computing 2000, Bristol, UK, Springer LNCS 1927, 172-186

[NCGG10] Nick T., Coersmeier E., Geldmacher J., Götze J. 2010. Classifying Means of Transportation Using Mobile Sensor Data. IEEE World Congress on Comp. Intelligence, Barcelona, Spain, July 2010

[ROPT05] Raento M., Oulasvirta A., Petit R., Toivonen H. 2005. ContextPhone: A Prototyping Platform for Context-Aware Mobile Applications. IEEE Pervasive Computing, April-June 2005, Vol. 4 No. 2, 51-59

[Ro08] Roth, J. 2008. A Probabilistic Approach for Context Reasoning. Use In Context, GI Informatik 2008, Munich, Germany, Sept. 12. 2008, Proceedings 134, Vol. 2, 802-807

[Ro11a] Roth, J. 2011. DateStone Calendar Manual. wireless-earth, http://android.wireless-earth.org/datestone.html

[Ro11b] Roth, J. 2011. Moving Geo Databases to Smart Phones – An Approach for Offline Location-based Applications. Innovative Internet Computing Systems (I^2CS), Berlin Germany, June 15-17 2011

[SHW+06] van Sinderen, M. J., van Halteren, A. T., Wegdam, M., Meeuwissen, H. B., Eertink, E. H. 2006. Supporting context-aware mobile applications: an infrastructure approach. IEEE Communications Magazine, Sept. 2006, Vol. 44, No. 9, 96-104

[Ve09] Verkasalo H. 2009. Contextual patterns in mobile service usage. Personal and Ubiquitous Computing. Vol. 13, No. 5, 331-342